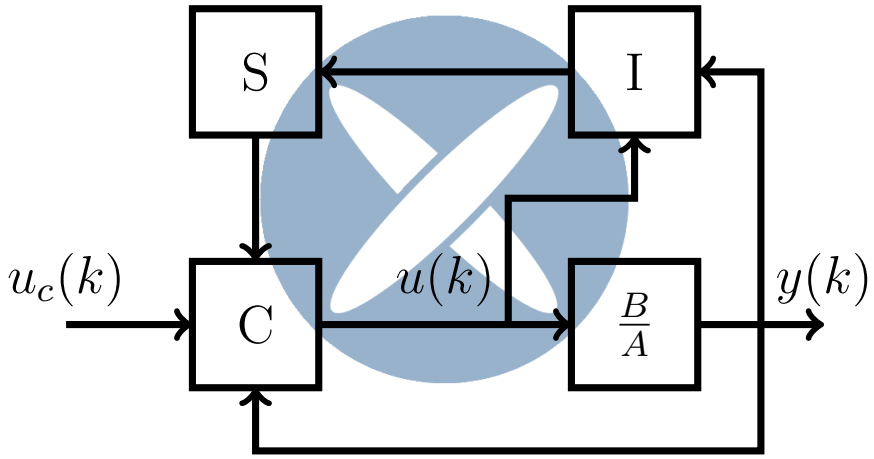
Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2012

Diego Armando Areias Mendes

Controlador Adaptativo Baseado em Xenomai Lab



Diego Armando Areias Mendes

Controlador Adaptativo Baseado em Xenomai Lab

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e de Telecomunicações, realizada sob a orientação científica de Dr. Alexandre Mota, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e Dr. Paulo Pedreiras, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática.

o júri / the jury

presidente / president

Professor Doutor José Alberto Gouveia Fonseca
Professor Associado, Universidade de Aveiro

vogais / examiners committee

Professor Doutor Alexandre Manuel Moutela Nunes da Mota
Professor Associado, Universidade de Aveiro (orientador)

Professor Doutor Paulo Bacelar Reis Pedreiras
Professor Auxiliar, Universidade de Aveiro (co-orientador)

Professor Doutor Luís Miguel Pinho de Almeida
Professor Associado, Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Eng. da Univ. do Porto

agradecimentos / acknowledgements

Gostaria de agradecer aos meus orientadores Prof. Doutor Alexandre Mota e Prof. Doutor Paulo Pedreiras pela ajuda e sugestões.

Ao Jorge Azevedo pela ajuda prestada na implementação de código em *Xenomai Lab*.

Ao Tiago Gonçalves pela ajuda que me deu por diversas vezes, entre elas o desenho da PCB e um pequeno "precalço" com um PIC32.

Ao César Gomes pela ajuda e *feedback* dado ao longo destes meses de trabalho.

À minha família pelo apoio e por me terem possibilitado seguir os estudos.

À minha namorada, Diana Borges, pelo apoio, paciência e compreensão.

Resumo

Um controlador adaptativo é um controlador que possui mecanismos que lhe permite adaptar-se a mudanças no comportamento do sistema a controlar. Esta dissertação descreve a implementação de um controlador adaptativo em *Xenomai Lab* e o desenvolvimento de uma plataforma de ensaio para o teste e validação do controlador em questão.

O trabalho desenvolvido terá aplicação como ferramenta de ensino da teoria de controlo ou como ferramenta de desenvolvimento de novos projetos.

Abstract

An adaptive controller is a controller equipped with mechanisms that allows it to adapt to the changes in the process it is meant to control. This dissertation describes the implementation of an adaptive controller on *Xenomai Lab* and the development of a test bed meant to test and validate said controller.

The work developed could be applied as an educational tool for control theory or as a development tool for new projects.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

No mundo em que vivemos muito do que nos rodeia depende de sistemas de controlo.

Por vezes, os sistemas a controlar apresentam parâmetros variantes no tempo. Um exemplo disso são os aviões a jato cuja a dinâmica depende da velocidade e altitude a que viajam. Outro exemplo de um sistema cujos parâmetros variam consideravelmente são os foguetões, cuja a massa diminui consideravelmente conforme se vai queimando o combustível [2].

Os controladores de parâmetros fixos possuem alguma tolerância em relação a variações do sistema. No entanto, a gama de tolerância é limitada e mesmo dentro dessa gama o comportamento do sistema em malha fechada poderá não ser o mais desejável podendo levar a perdas de eficiência.

Uma abordagem mais adequada para lidar com sistemas variantes no tempo são os controladores adaptativos. Um controlador adaptativo é um controlador que possui mecanismos que lhe permite adaptar-se a mudanças no comportamento do sistema a controlar. Este tipo de controladores possui duas malhas: uma de realimentação para controlo tal como os controladores de parâmetros fixos e outra malha para o ajuste dos parâmetros do controlador. Os mecanismos presentes na malha de ajuste de parâmetros são responsáveis pela identificação do modelo do sistema a controlar e pelo projeto de um novo controlador [2].

Atualmente, a maioria dos controladores é implementada em eletrónica digital. A maioria das técnicas de controlo digital estão assentes no pressuposto de que o período de amostragem é preciso e constante. O desrespeito por este pressuposto poderá significar uma performance reduzida ou até instabilidade do sistema de controlo.

Consequentemente, os sistemas de controlo digitais são considerados sistemas de tempo real. Um sistema de tempo-real é um sistema de computação em que a exatidão dos resultados não depende apenas do resultado lógico, depende também do momento em que esses resultados são produzidos [3].

A implementação de um sistema de tempo real num *personal computer* (PC) requer um sistema operativo de tempo-real. Um sistema operativo de tempo-real é uma plataforma que permite a execução de processos com requisitos temporais restritos.

Alguns dos sistemas operativos de tempo real mais conhecidos são o *LynxOS* [4], *QNX* [5] ou o *VxWorks* [6]. O *Xenomai* [7], por sua vez, é uma das modificações existentes para o *kernel* de *Linux* que permitem melhorar a performance de tempo-real de um sistema operativo baseado em *Linux*.

No âmbito da sua dissertação de mestrado, Jorge Azevedo desenvolveu para *Xenomai* a aplicação *Xenomai Lab* [8, 9]. Esta aplicação, na linha de aplicações como o *Simulink*

[10] ou *Xcos* [11], possibilita o projeto e implementação de diagramas de blocos capazes de representar sistemas de controlo. O utilizador poderá utilizar a aplicação para apenas simular um sistema de controlo ou utilizar as portas de I/O do computador para efetivamente controlar um sistema físico [8].

Dada a facilidade de utilização do *Xenomai Lab*, esta será a plataforma na qual se implementará o controlador a desenvolver.

1.1 Objetivos

Esta dissertação tem dois objetivos principais: desenvolver e implementar em *Xenomai Lab* um controlador adaptativo baseado em posicionamento de polos e desenvolver uma plataforma de ensaio para o teste e validação do controlador em questão.

A técnica de posicionamento de polos é uma técnica de controlo moderno que procura garantir que a equação característica de um sistema de controlo em malha fechada seja exatamente igual à equação característica definida.

A plataforma de ensaio será constituída por uma interface analógica-digital e digital-analógica (A-D e D-A) para um PC e um simulador de processos contínuos.

O trabalho desenvolvido poderá ter aplicação como ferramenta de ensino de teoria de controlo ou como ferramenta de desenvolvimento de novos projetos.

1.2 Estrutura da dissertação

Os conceitos fundamentais para o desenvolvimento do trabalho no âmbito desta dissertação são abordados nos capítulos 2 e 3. No capítulo 2 são abordados os conceitos fundamentais de controlo de tempo-real, incluindo conceitos de controlo digital e conceitos de sistemas de tempo-real. No capítulo 3 são abordados tópicos de controlo adaptativo, incluindo a identificação de sistemas e o projeto do controlador.

No capítulo 4 são apresentados e analisados diversos algoritmos úteis para controlo adaptativo.

O desenvolvimento da plataforma de ensaio é abordado nos capítulos 5 e 6. No capítulo 5 é abordado o projeto, desenvolvimento e validação da interface A-D e D-A. No capítulo 6 é apresentado o projeto, desenvolvimento e validação do simulador de processos contínuos.

No capítulo 7 são apresentados os resultados de alguns testes realizados ao controlador adaptativo implementado em *Xenomai Lab*. Para todos os testes realizados é utilizada a plataforma de ensaio apresentada nos capítulos 5 e 6.

No capítulo 8 são apresentadas as conclusões do trabalho desenvolvido e algumas sugestões para trabalho futuro.

Nos apêndices são apresentados alguns complementos de interesse ao conteúdo apresentado ao longo desta dissertação.

Capítulo 2

Conceitos fundamentais sobre controlo de tempo-real

Este capítulo procura descrever alguns conceitos fundamentais sobre controlo de tempo-real.

Numa primeira parte serão abordados tópicos de controlo digital. Será explicada a necessidade de uma interface entre sistemas de natureza diferente (contínuos e discretos). Será também discutido o modelo matemático de um sistema discreto, incluindo a transformada z.

Na segunda parte deste capítulo serão abordados os sistemas de tempo-real. A abordagem deste tema incidirá sobre as restrições temporais destes sistemas, algoritmos de escalonamento e as implicações da não utilização deste tipo de sistemas em controlo digital.

2.1 Tópicos de controlo digital

Os controladores analógicos eram, até há umas décadas, a única opção disponível para controlar processos físicos. No entanto, nas últimas décadas os controladores digitais tornaram-se os controladores de eleição.

A preferência por este tipo de controladores deve-se à sua fiabilidade, custo reduzido e flexibilidade. Fiáveis pois são consideravelmente mais robustos e menos sensíveis a variações do ambiente em que estão inseridos do que os controladores analógicos. O custo reduzido atual é fruto da massificação dos microprocessadores e do crescente número de profissionais na área de controlo digital. Flexíveis porque, em caso de ser necessário algum ajuste no controlador, é suficiente reprogramar o algoritmo que é suposto executar, enquanto que um controlador analógico poderá ficar logo inutilizável na mesma situação [12].

No entanto, dois sistemas de natureza distinta, como é o caso do sistema a controlar analógico e o controlador digital, não podem interagir diretamente. Existe a necessidade de uma interface capaz de converter os sinais entre ambos os domínios.

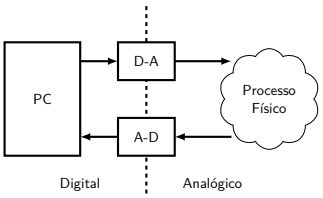


Figura 2.1: Interface entre domínios diferentes

2.1.1 Interfaces A-D e D-A

Uma interface apropriada entre um sistema analógico e um digital é composta por conversores analógicos-digitais (A-D) e conversores digitais-analógicos (D-A).

Na figura 2.2 é apresentado um esquema simplificado de um sistema de controlo digital. No esquema são, também, apresentados os diversos sinais presentes num sistema de controlo digital, bem como os processos que os relacionam.

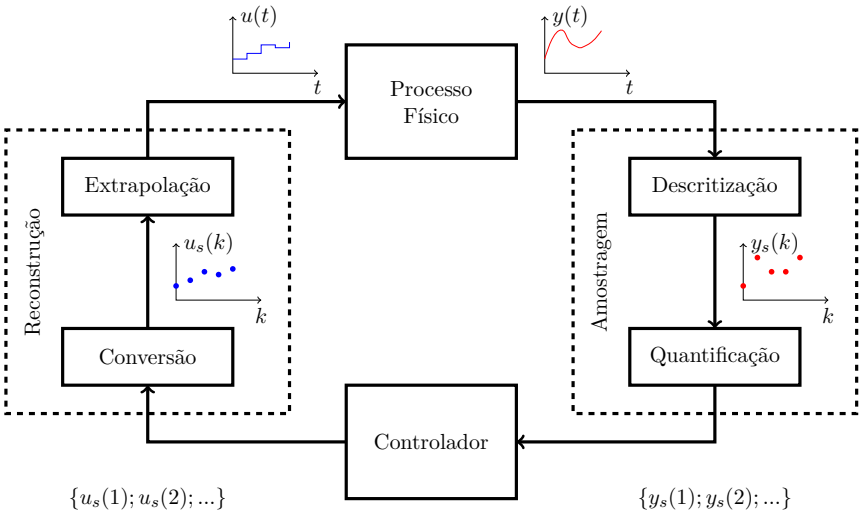


Figura 2.2: Relação entre o sinal amostrado ($y(t)$), sinal de controlo ($u(t)$) e as representações correspondentes no sistema computacional.

O processo de amostragem é da responsabilidade do conversor A-D. Neste processo um sinal analógico do processo físico é convertido num sinal digital que o controlador seja capaz de processar.

Por sua vez, o processo de reconstrução é da responsabilidade do conversor D-A. Durante este processo o sinal digital produzido pelo controlador é convertido num sinal analógico que possa ser aplicado ao processo físico.

2.1.1.1 Amostragem

A amostragem de um sinal contínuo consiste na sua substituição por uma sequência de números que representam o seu valor em determinados instantes [1].

Segundo Buttazzo [13], o processo de amostragem pode ser decomposto nos processos de discretização e quantificação. A discretização consiste na obtenção de uma amostra do

sinal contínuo em instantes discretos de tempo. O intervalo de tempo entre estes instante é, na maioria dos casos, constante e é conhecido como período de amostragem. A quantificação consiste na conversão do valor da amostra obtida durante a discretização no valor mais próximo pertencente a um conjunto finito de valores [14].

É necessário ter em atenção que o sinal digital obtido possui uma resolução finita, fruto da quantificação. A resolução é uma das características de um conversor A-D. Se um conversor A-D possuir n bits e admitir na sua entrada sinais analógicos que variem entre V_{min} e V_{max}, então a sua resolução, ou nível de quantificação, Q é dada por:

$$Q = \frac{V_{max} - V_{min}}{2^n - 1} \tag{2.1}$$

Isto significa que o sinal amostrado possuirá sempre um erro de quantificação e_q cujo módulo da amplitude é restringido por:

$$\|e_q\| \leq \frac{1}{2}Q \tag{2.2}$$

Caso se pretenda diminuir o majorante do módulo do erro de quantificação, deve-se escolher um conversor A-D com um nível de quantificação menor.

2.1.1.2 Reconstrução

A reconstrução de um sinal analógico é o processo inverso à amostragem, convertendo um sinal digital, uma sequência de números, num sinal analógico contínuo no tempo.

Este processo também pode ser decomposto em dois processos mais simples: conversão e extrapolação. Durante a conversão a sequência de números é convertida numa sequência de sinais elétricos de amplitude proporcional ao número que representam. O processo de extrapolação consiste na geração de uma sinal contínuo no tempo a partir dos valores dos sinais gerados pela conversão.

Na figura 2.2 o sinal analógico $u(t)$ é reconstruído usando um extrapolador de ordem zero, conhecido como *zero-order hold*.

2.1.1.3 Frequência de amostragem

Outro fator determinante no funcionamento de um sistema de controlo digital é a frequência de amostragem.

O teorema de amostragem de Nyquist-Shannon diz que para que um sinal amostrado possa ser devidamente reconstruído, a frequência de amostragem deverá ser superior ao dobro da maior frequência do sinal. Na prática, este resultado é necessário mas não suficiente para definir um período de amostragem adequado a um sistema de controlo.

Uma técnica adequada para definir o período de amostragem h consiste em dividir o tempo de subida do sistema a controlar T_r pelo número de amostras por tempo de subida N_r (equação 2.3). Segundo Åström and Wittenmark [1], para que se obtenha uma performance aceitável do sistema de controlo deve atribuir-se a N_r um valor entre 4 e 10.

$$h = \frac{T_r}{N_r} \tag{2.3}$$

É também necessário ter em conta o fenómeno conhecido por *aliasing* ou *frequency folding* quando se define a frequência de amostragem.

O *aliasing* é um fenómeno em que componentes de frequências que não existiam no sinal a amostrar são geradas no sinal amostrado devido à natureza do processo de amostragem. Pode ser provocado por um mau acondicionamento do sinal a amostrar ou pela escolha de uma frequência de amostragem desadequada.

O comportamento dos conversores A-D e conversores D-A, bem como o fenómeno de *aliasing* encontram-se devidamente documentados em Proakis and Manolakis [14], Ogata [12], Åström and Wittenmark [1].

2.1.2 O modelo discreto

O modelo discreto é o modelo matemático que descreve o comportamento de um sistema dinâmico discreto.

Graças à utilização de interfaces A-D e D-A é possível obter-se um sistema discreto a partir de um sistema contínuo. O modelo matemático deste sistema discreto é produto do modelo da interface e do modelo do sistema contínuo. Este modelo discreto é também dependente da frequência de amostragem.

O comportamento de um sistema dinâmico linear no domínio discreto pode ser descrito nos instantes de amostragem pelas equações de diferença.

$$y(k+n)+a_1y(k+n-1)+...+a_{n-1}y(k+1)+a_ny(k) = b_0u(k+m)+...+b_{m-1}u(k+1)+b_mu(k) \tag{2.4}$$

No entanto, uma representação mais vantajosa é a que utiliza a transformada z. De certa forma, a transformada z está para os sistemas discretos como a transformada de Laplace está para os sistemas contínuos [1].

A transformada z introduz a variável complexa z, e é dada pela equação 2.5.

$$F(z) = \sum_{k=-\infty}^{\infty} f(kh)z^{-k} \tag{2.5}$$

Uma das propriedades de maior interesse da transformada z para a análise de sistemas discretos é a de translação no tempo, ou seja, $y(k+n) \overset{z}{\Longleftrightarrow} z^nY(z)$. Isto significa que a equação 2.4 pode ser reescrita como se apresenta na equação 2.6.

$$z^nY(z)+a_1z^{n-1}Y(z)+...+a_{n-1}zY(z)+a_nY(z) = b_0z^mU(z)+...+b_{m-1}zU(z)+b_mU(z) \tag{2.6}$$

A utilização da transformada z permite a obtenção da função de transferência do sistema e, consequentemente, a sua análise num domínio complexo de frequência (domínio de z).

A análise no domínio de z consiste em determinar o comportamento do sistema através da localização dos seus polos e zeros no plano complexo. Este tipo de análise é análoga à análise no domínio de Laplace para os sistemas contínuos.

A figura 2.3 apresenta uma comparação entre o domínio de z e o domínio de Laplace. A relação entre as variáveis complexas da transformada z e transformada de Laplace, z e s respetivamente, é dada por $z = e^{sh}$, sendo h o período de amostragem.

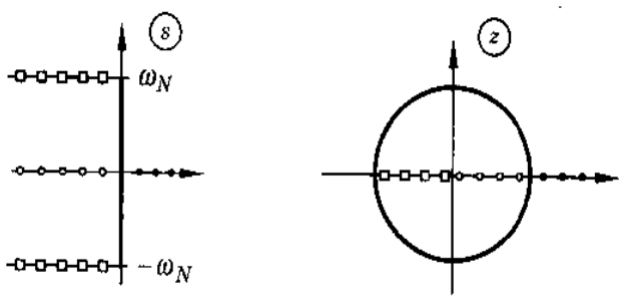


Figura 2.3: Domínio de Laplace (esquerda) e domínio de z (direita)[1]

De notar que, todo o semi-plano esquerdo do domínio de Laplace fica contido dentro do círculo de raio unitário centrado na origem do domínio de z. Assim, é condição suficiente para a estabilidade de um sistema discreto que todos os polos do sistema se encontrem dentro do círculo de raio unitário no plano complexo da transformada z.

2.2 Tópicos de sistemas de tempo-real

Um sistema de tempo-real é um sistema computacional que precisa reagir a eventos num ambiente externo respeitando restrições temporais precisas [13].

Segundo Stankovic [3], num sistema de computação de tempo-real a exatidão dos resultados não depende apenas do resultado lógico, depende também do momento em que esses resultados são produzidos.

Uma confusão comum é a de que um sistema rápido é o equivalente a um sistema de tempo-real. No entanto, a previsibilidade dos tempos de resposta é na verdade a propriedade mais importante para os sistemas de tempo-real, pois a rapidez só por si apenas garante que os tempos de resposta médios são minimizados [3].

Neste capítulo os tópicos de sistemas de tempo-real serão apenas introduzidos de forma abreviada. Para uma discussão mais detalhada do assunto sugere-se a leitura de Buttazzo [13].

2.2.1 Requisitos e restrições

Um sistema de tempo-real possui requisitos funcionais e requisitos não funcionais, compostos por requisitos temporais e requisitos de dependabilidade.

Os requisitos funcionais referem-se à necessidade dos resultados produzidos pelo sistema serem logicamente corretos.

Os requisitos temporais referem-se à necessidade dos resultados do sistema serem produzidos enquanto são úteis.

Os requisitos de dependabilidade referem-se à necessidade de um sistema de tempo-real possuir probabilidade de falhas extremamente baixa e advêm do facto deste tipo de sistemas serem normalmente usados em aplicações críticas.

Num sistema de tempo-real as aplicações executadas respeitam um modelo computacional conhecido como modelo de tempo-real. Um modelo computacional de tempo-real é um modelo segundo o qual a aplicação executa indefinidamente uma sequência de iterações, processando um fluxo de dados com o qual está sincronizada e que lhe impõe restrições temporais.

As restrições temporais dependem da aplicação em que o sistema é utilizado, sendo classificadas de acordo com a utilidade dos resultados produzidos caso não sejam respeitadas. As restrições temporais são classificadas como [13]:

- Suave (*Soft*) - Com este tipo de restrições os resultados produzidos possuem sempre alguma utilidade, a qual se vai reduzindo gradualmente após a *deadline* ser ultrapassada.

- Firme (*Firm*) - Quando este tipo de restrição não é respeitada o resultado produzido perde qualquer utilidade.

- Rígida (*Hard*) - Quando uma restrição temporal deste tipo não é respeitada, pode dar-se uma falha catastrófica no sistema (p.e. perdas materiais de valor elevado ou mesmo de vida humana).

Em função das classificações das restrições temporais, o sistema também pode ser classificado, neste caso como:

- Sistema de tempo-real rígido (*Hard Real-Time*) - É um sistema que possui pelo menos uma restrição temporal rígida.

- Sistema de tempo-real suave (*Soft Real-Time*) - É um sistema que apenas possui restrições temporais suaves e firmes.

2.2.2 Escalonamento

Num sistema de tempo-real podem ser executados múltiplas aplicações, sendo cada uma destas composta por múltiplas tarefas. Assim, existe a necessidade de gerir o acesso das tarefas ao CPU bem como a recursos partilhados.

Essa gestão é responsabilidade do algoritmo de escalonamento. O algoritmo de escalonamento determina qual é a melhor ordem de execução das tarefas do sistema a partir da informação que dispõem sobre estas (precedências, tempo de execução no pior caso, *deadline*, recursos necessários, prioridade, etc...).

A ordem de execução das tarefas também precisa de garantir que o fluxo correto das aplicações é respeitado, para que estas possam produzir resultados válidos.

Segundo Buttazzo [13], os algoritmos de escalonamento podem ser classificados como:

- preemptivo ou não-preemptivo - Preemptivo caso haja suporte do sistema à interrupção de tarefas em execução para atribuição do CPU a outras tarefas. Não-preemptivo quando todas as tarefas são executadas sem serem interrompidas.

- Baseado em prioridades Estáticas ou dinâmicas - De acordo com a forma como as prioridades são atribuídas às tarefas. Estático caso as prioridades sejam fixas e definidas antes da ativação do sistema, dinâmico caso contrário.

- *On-line* ou *off-line* - *Off-line* quando toda a ordem de execução de tarefas é definida antes da ativação de qualquer tarefa. *On-line* quando a ordem de execução das tarefas é definida em paralelo com a execução das mesmas. Estes algoritmos são necessários quando há tarefas que são ativadas ou terminadas durante a operação do sistema.

- Ótimo ou sub-ótimo (heurístico) - Um algoritmo de escalonamento é ótimo quando minimiza uma função de custo. Do ponto de vista de escalonamento, um algoritmo diz-se ótimo, dentro de uma dada classe, se consegue encontrar um escalonamento praticável sempre que qualquer outro algoritmo da mesma classe é também capaz de encontrar um escalonamento praticável. Um algoritmo de escalonamento é sub-ótimo quando pode determinar um escalonamento ótimo, mas não existe garantias de o conseguir.

Exemplos de algoritmos de escalonamento são o *Rate Monotonic* (RM), ótimo entre os algoritmos de prioridades fixas, ou o *Earliest Deadline First* (EDF), ótimo entre os algoritmos de prioridades dinâmicas.

2.2.3 Sistemas de tempo-real e controlo

Nos sistemas de controlo de tempo-real as restrições temporais são derivadas da dinâmica do processo com o qual o sistema interage. Normalmente, as restrições temporais aplicam-se aos atrasos de observação do sistema, aos atrasos de atuação e às variações desses atrasos (*jitter*).

Supondo que o leitor está familiarizado com a teoria de sistemas, apresenta-se na figura 2.4 um modelo em que o atraso de observação é equiparado a um atraso Δt na malha de realimentação. Por sua vez, a equação 2.7 apresenta a função de transferência para o sistema em malha fechada.

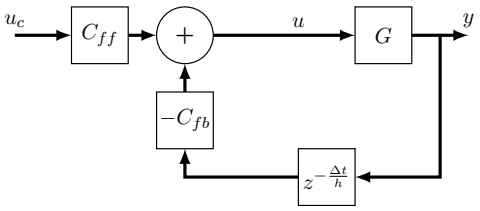


Figura 2.4: Atraso de amostragem

$$\frac{Y(z)}{U_c(z)} = \frac{G(z)C_{ff}(z)}{1 + G(z)C_{fb}(z)z^{-\frac{\Delta t}{h}}} \tag{2.7}$$

Da análise da equação 2.7 podemos concluir que o atraso de observação afeta o posicionamento dos polos do sistema em malha fechada, podendo afetar a sua estabilidade.

Na figura 2.5 é apresentado um modelo em que o atraso de atuação é equiparado a um atraso Δt entre o controlador e o sistema a controlar. A equação 2.8 demonstra também o efeito do atraso na função de transferência do sistema em malha fechada.

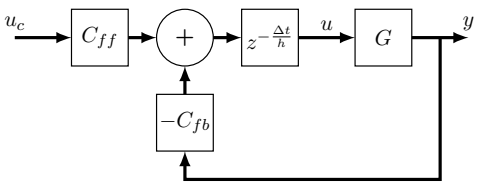


Figura 2.5: Atraso de atuação

$$\frac{Y(z)}{U_c(z)} = \frac{G(z)z^{-\frac{\Delta t}{h}}C_{ff}(z)}{1 + G(z)z^{-\frac{\Delta t}{h}}C_{fb}(z)} \tag{2.8}$$

Na equação 2.8 é possível constatar que o efeito do atraso de atuação é equivalente a que o sistema a controlar G possua um tempo morto Δt. Este atraso resulta num sistema em malha fechada com tempo morto e com um deslocamento dos polos.

O efeito do *jitter* dos atrasos de observação e atuação é talvez o efeito mais indesejável dada a sua imprevisibilidade. Os erros introduzidos no sistema acabam por ser equivalentes a ter um sistema com parâmetros e tempo morto variável.

Na figura 2.6 podemos ver um exemplo de um sinal amostrado com período constante e que sofre *jitter* ao efetuar uma amostra.

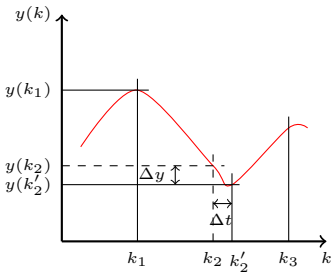


Figura 2.6: Sinal amostrado com *jitter*

O erro na medida Δy é dado por:

$$\Delta y = \int_{k_2}^{k_2+\Delta t} y'(t)dt \tag{2.9}$$

Assim, $|\Delta y|$ será, potencialmente, tanto maior quanto maior for $|y'|$ ou quanto maior for $|\Delta t|$. Um estudo sobre os efeitos de *jitter* de amostragem no desempenho de alguns tipos de controladores foi realizado por Yu et al. [15], os resultados obtidos demonstram que controladores com componentes derivativas são especialmente sensíveis ao efeito de *jitter* de amostragem.

Capítulo 3

Controlo adaptativo baseado em posicionamento de polos

A teoria de controlo adaptativo procura o desenvolvimento de controladores adequados a sistemas variantes no tempo ou sistemas que apesar de serem invariantes no tempo possuem parâmetros desconhecidos. De acordo com Åström and Wittenmark [2], um controlador adaptativo é um conjunto composto por um controlador de parâmetros ajustáveis e um mecanismo de ajuste desses parâmetros.

O controlo adaptativo encontra aplicações em áreas como a indústria aeroespacial, controlo de processos industriais, navegação assistida de embarcações marítimas, robótica, indústria automóvel e indústria biomédica. Algumas das aplicações mais frequentes são o ajuste automático de controladores (*automatic tuning*), escalonamento de ganho (*gain scheduling*) e adaptação contínua (*continuous adaptation*) [2].

A adaptação contínua é a única aplicação de interesse no âmbito desta dissertação. É uma técnica assente na identificação em tempo-real dos parâmetros do processo permitindo o ajuste também, em tempo-real, dos parâmetros do controlador. Este tipo de adaptação poderá ser efetuada através do método direto ou do método indireto [1]. O método direto deduz os parâmetros adequados para o controlador a partir da resposta do processo aos seus sinais de entrada. Já o método indireto começa por identificar o modelo matemático do processo e só depois procede ao cálculo dos parâmetros do controlador.

Neste capítulo é apresentada a implementação do sistema de controlo desenvolvido nesta dissertação, bem como alguns conceitos básicos para a sua compreensão. O sistema, como pode ser visto na figura 3.1, é composto por quatro blocos. O bloco $\frac{B}{A}$ representa o sistema a controlar. O bloco C é o controlador do sistema. O bloco S é o bloco que calcula os parâmetros do controlador. E por fim o bloco I é o bloco de identificação do modelo do sistema a controlar.

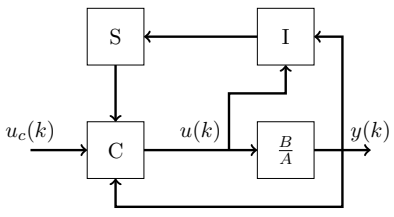


Figura 3.1: Sistema de controlo adaptativo

Este sistema de controlo procura que um sistema dinâmico de parâmetros desconhecidos, equação 3.1, se comporte como um sistema cujos parâmetros são dados pelos polinómios B_m e A_m, equação 3.2.

$$A(z^{-1})y(k) = B(z^{-1})u(k) \tag{3.1}$$

$$A_m(z^{-1})y(k) = B_m(z^{-1})u_c(k) \tag{3.2}$$

Segue-se agora uma breve descrição sobre os objetivos e funcionamento dos blocos C, S e I.

3.1 Controlador

Na equação 3.3 apresenta-se a representação polinomial de um controlador linear genérico com dois graus de liberdade no domínio discreto.

Esta é a representação usada para a implementação do controlador nesta dissertação. Os polinómios R, S e T são os polinómios que o caracterizam e são determinados pelo bloco de síntese de controlador. Os sinais do sistema são o sinal de controlo u, o sinal de referência u_c e o sinal de saída y.

$$R(z^{-1})u(k) = T(z^{-1})u_c(k) - S(z^{-1})y(k) \tag{3.3}$$

O controlador calcula o novo valor para o sinal u em função de amostras atuais e/ou anteriores dos diversos sinais e dos coeficientes dos polinómios. A componente de realimentação tem como função de transferência $-\frac{S}{R}$ e a componente direta possui a função de transferência $\frac{T}{R}$.

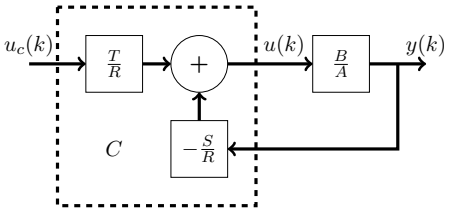


Figura 3.2: Controlador RST

3.2 Síntese de controlador

O objetivo deste bloco é determinar os polinómios R, S e T tais que o controlador da equação 3.3 permita obter o sistema em malha fechada desejado. As equações que se seguem aplicam-se apenas ao projeto de um controlador que não cancela zeros (os zeros do sistema em malha aberta são mantidos em malha fechada).

Das equações 3.1 e 3.3 deduz-se que o sistema em malha fechada é dado por:

$$y(k) = \frac{B(z^{-1})T(z^{-1})}{A(z^{-1})R(z^{-1}) + B(z^{-1})S(z^{-1})}u_c(k) \tag{3.4}$$

Identifique-se, então, o polinómio característico do sistema em malha fechada como A_c:

$$A_c = AR + BS \tag{3.5}$$

A equação 3.5 é o núcleo deste bloco. Este tipo de equação encontra-se em diversos problemas algébricos, sendo conhecida pelos nomes de equação de Diophantine, identidade de Bezout ou identidade de Aryabhatta [2]. A sua resolução permite obter os polinómios desconhecidos R e S a partir dos polinómios conhecidos A, B e A_c. Como permite colocar os polos do sistema em malha fechada onde o projetista especificar, a resolução desta equação pode ser considerada como uma técnica de posicionamento de polos.

Com vista à implementação do controlador considere-se que os fatores desejados para o polinómio R são representados por R_d e os fatores desejados para o polinómio S são representados por S_d. A equação 3.5 pode ser reescrita conforme se apresenta na equação 3.6, os polinómios $A' = AR_d$ e $B' = BS_d$ são os novos polinómios conhecidos e os polinómios R' e S' são então os polinómios desconhecidos.

$$A_c = AR_dR' + BS_dS' = A'R' + B'S' \tag{3.6}$$

A título de exemplo, um controlador com $R_d = z - 1$ possuirá ação integral e um controlador com $S_d = z + 1$ será capaz de filtrar o ruído de alta-frequência na malha de realimentação [1].

A equação 3.4 leva a inferir que poderão haver fatores cancelados entre BT e A_c. Considerando o polinómio A_m como sendo um fator do polinómio A_c, apresenta-se a equação 3.7. Nesta equação é também introduzido o polinómio observador A_o. O polinómio A_o é arbitrário servindo para que A_c seja de ordem suficientemente elevada para garantir uma solução para a equação 3.6. O efeito do polinómio A_o na transmissão de perturbações e ruído está documentado em Åström and Wittenmark [1].

$$A_c = A_mA_o \tag{3.7}$$

Apesar de essencial à síntese do controlador o polinómio A_o não faz parte do sistema em malha fechada. É então necessário que o seu efeito seja cancelado. Como o polinómio A_o é um fator do denominador da equação 3.4, a solução é introduzi-lo como um fator do numerador. Assim, o polinómio T do controlador é dado pela equação 3.8.

$$T = \frac{A_m(1)}{B(1)}A_o \tag{3.8}$$

O projeto do controlador necessita também de respeitar algumas condições, não só de forma a garantir que é possível a resolução dos cálculos envolvidos como também garantir a causalidade do mesmo.

Se analisarmos a equação 3.3 concluí-se que para o controlador ser causal é necessário garantir que:

$$deg(S) \leq deg(R) \tag{3.9}$$

$$deg(T) \leq deg(R) \tag{3.10}$$

em que $deg()$ corresponde à ordem do polinómio.

Considerando que o sistema a controlar é causal, temos $deg(A) \geq deg(B)$. Se tivermos também em conta a equação 3.9 podemos considerar que AR é o termo de maior ordem de A_c e como tal:

$$deg(R) = deg(A_c) - deg(A) \tag{3.11}$$

A condição seguinte é então resultado das equações 3.8, 3.10 e 3.11. O número de polos em excesso do sistema desejado nunca pode ser inferior ao número de polos em excesso do sistema a controlar.

$$deg(A) - deg(B) \leq deg(A_m) - deg(B_m) \tag{3.12}$$

Agora, se considerarmos que R'_0 e S'_0 são soluções para a equação 3.6, estas não são soluções únicas. O conjunto de soluções possíveis é dado pela equação 3.13, em que Q é um polinómio arbitrário.

$$\begin{aligned} R' &= R'_0 + QB' \\ S' &= S'_0 - QA' \end{aligned} \tag{3.13}$$

Isto significa que para a solução de ordem mínima o polinómio S' apresentará, no máximo, uma ordem igual a $deg(A') - 1$. A existência de uma solução única para o controlador de ordem mínima é comprovada por [16].

$$deg(S') \leq deg(A') - 1 \tag{3.14}$$

Ou seja, partindo do princípio de que $deg(S') = deg(A) + deg(R_d) - 1$, temos a garantia de que existe uma solução. O controlador de menor ordem que garante a causalidade será tal que $deg(S) = deg(T) = deg(R)$, consequentemente:

$$deg(S') = deg(A) + deg(R_d) - 1 \tag{3.15}$$

$$deg(R') = deg(A) + deg(S_d) - 1 \tag{3.16}$$

A análise da equações 3.7, 3.11 e 3.16 leva-nos então a concluir que uma condição válida para o dimensionamento do polinómio A_o para o controlador de ordem mínima será:

$$deg(A_o) = 2deg(A) + deg(R_d) + deg(S_d) - deg(A_m) - 1 \tag{3.17}$$

3.3 Identificação de sistemas

Este bloco estima o modelo matemático de um sistema dinâmico a partir dos dados recolhidos. A qualidade das estimativas obtidas pelos algoritmos utilizados é avaliada de acordo com um dado critério.

Um modelo matemático é uma representação, por via de expressões matemáticas, dos aspetos essenciais de um sistema com a capacidade de descrever o seu comportamento.

Todos os modelos polinomiais de sistemas lineares são casos particulares de um modelo conhecido como modelo geral. O modelo geral é dado pela equação 3.18. A parte determinística do modelo é "estimulada" pelo sinal u e possui a função de transferência $\frac{B}{AF}$. Já a parte estocástica do modelo é "estimulada" pelo sinal e, perturbações do sistema, e possui a função de transferência $\frac{C}{AD}$.

$$A(z^{-1})y(k) = z^{-d}\frac{B(z^{-1})}{F(z^{-1})}u(k) + \frac{C(z^{-1})}{D(z^{-1})}e(k) \tag{3.18}$$

O modelo ARMAX (*autoregressive-moving average with exogenous terms*) surge como uma simplificação do modelo geral, com $F = D = 1$, e é dado pela equação 3.19. Este modelo é apropriado para sistemas cujas perturbações atuem essencialmente na sua entrada. Isto porque a parte determinística e a estocástica possuem o mesmo conjunto de polos [17].

$$A(z^{-1})y(k) = z^{-d}B(z^{-1})u(k) + C(z^{-1})e(k) \tag{3.19}$$

O modelo ARX (*autoregressive with exogenous terms*), equação 3.20, é um simplificação do modelo ARMAX em que $C = 1$. Esta simplificação resulta num modelo com menor flexibilidade para lidar com perturbações. Deve, por este facto, apenas ser utilizado quando as perturbações atuem à entrada do sistema e sejam ruído branco com média nula.

$$A(z^{-1})y(k) = z^{-d}B(z^{-1})u(k) + e(k) \tag{3.20}$$

Para perturbações que atuem essencialmente à saída do sistema, p.e. interferências nas medições, existe o modelo Box-Jenkins. Este modelo, apresentado na equação 3.21, é uma simplificação do modelo geral em que $A = 1$.

$$y(k) = z^{-d}\frac{B(z^{-1})}{F(z^{-1})}u(k) + \frac{C(z^{-1})}{D(z^{-1})}e(k) \tag{3.21}$$

O modelo OE (*Output Error*), apresentado na equação 3.22, é uma simplificação do modelo Box-Jenkins em que $C = D = 1$. Tal como o modelo Box-Jenkins, este modelo é indicado para sistemas em que as perturbações atuem à saída. No entanto, como não modela a parte estocástica, este sistema deverá apenas ser usado quando as perturbações são ruído branco com média nula.

$$y(k) = z^{-d}\frac{B(z^{-1})}{F(z^{-1})}u(k) + e(k) \tag{3.22}$$

Para a estimação dos coeficientes dos polinómios que compõem os modelos expostos existem diferentes algoritmos. Os diferentes algoritmos podem ser catalogados conforme as suas características. Alguns são de implementação recursiva (*on-line*) e analisam continuamente o comportamento de um sistema, noutros a estimação é feita a partir de um conjunto limitado de dados recolhidos previamente (*off-line*). Existem ainda algoritmos que analisam tanto as entradas como as saídas do sistema, contrastando com outros que apenas analisam a saída (p.e. *Frequency domain decomposition* [18]).

Apesar da diversidade de algoritmos de identificação existentes, o foco desta dissertação são essencialmente os algoritmos de implementação *on-line* em que se analisam as entradas e as saídas do sistema.

Ainda que, por vezes, apresentem resultados menos precisos que as implementações *off-line*, as implementações *on-line* recorrem a algoritmos recursivos e por isso podem ser parte de uma malha de controlo de tempo-real. A implementação recursiva permite obter um melhor modelo do sistema a controlar a cada iteração do ciclo de controlo.

Alguns exemplos de algoritmos para implementação *on-line* de identificação de sistemas são o *Generalized Least Squares* (GLS), o *Recursive Maximum Likelihood* (RML) e o *Recursive Least Squares* (RLS). Nesta dissertação apenas será tratado o algoritmo RLS e algumas das suas variações. Entre essas variações encontra-se o algoritmo *Recursive Extended Least Squares* (RELS) ou *Recursive Maximum Likelihood of the First Kind* (RML_1), que não é mais que uma extensão de RLS capaz de estimar um modelo ARMAX.

Existem também algumas simplificações do RLS, que apesar de oferecerem resultados menos precisos podem ser implementados em sistemas com menor capacidade de processamento. Entre essas simplificações encontram-se os algoritmos *Least Mean Squares* (LMS), *Projection Algorithm* (PA) e *Stochastic Approximation* (SA) [2].

Como já foi referido, os vários métodos necessitam de um critério para poderem avaliar os modelos estimados. Entre os critérios existentes destacam-se o critério de máxima verosimilhança e o de mínimos quadrados.

O critério de mínimos quadrados diz que o melhor resultado para uma estimativa é obtido para o mínimo absoluto da função de custo, equação3.23.

$$J(\theta) = \frac{1}{2}\sum_{k=1}^{N}(y_k - \hat{y}_k)^2 \tag{3.23}$$

Já o critério de máxima verosimilhança diz que o melhor resultado é obtido para o máximo absoluto da função de verosimilhança, equação 3.24. A função f é a função densidade de probabilidade conjunta.

$$L(\theta|y_1, ..., y_N) = \prod_{k=1}^{N} f(y_k|\theta) \tag{3.24}$$

Dos dois critérios, o critério de mínimos quadrados é o mais simples e é também aquele para o qual mais facilmente se deduz uma implementação recursiva.

Capítulo 4

Algoritmos de identificação de sistemas e algoritmos para a resolução da equação de Diophantine

Neste capítulo são apresentados e avaliados diversos algoritmos de interesse para um sistema de controlo adaptativo.

Na primeira parte são apresentados algoritmos de identificação de sistemas. É apresentado o método de mínimos quadrados nas suas vertentes não-recursiva e recursiva. São também apresentados algoritmos derivados da implementação recursiva do método de mínimos quadrados.

Na segunda parte são apresentados algoritmos para a resolução da equação de Diophantine pela via polinomial e via matricial.

A terceira parte é composta por alguns exemplos de implementação dos algoritmos. São também realizadas algumas análises aos mesmos algoritmos.

Na quarta parte é apresentado um exemplo de um sistema de controlo adaptativo. Com o objetivo de validar a implementação dos algoritmos em *Xenomai Lab*, a simulação do sistema de controlo é efetuada em *Xenomai Lab* e *MATLAB* e os resultados das simulações são comparados.

4.1 Algoritmos de identificação de sistemas

4.1.1 O método de mínimos quadrados

O método dos mínimos quadrados pressupõe que uma função $y(k)$ possa ser aproximada por:

$$\hat{y}(k) = \theta_0 + \varphi_1(k)\theta_1 + ... + \varphi_n(k)\theta_n \tag{4.1}$$

Em que φ_i são funções conhecidas e θ_i são parâmetros desconhecidos. As funções conhecidas φ_i, que poderão ser dependentes entre elas, são conhecidas como regressores. A equação 4.1 é chamada modelo de regressão.

Partindo do pressuposto de que todas as medições são efetuadas com a mesma precisão, o conjunto de parâmetros desconhecidos θ que melhor representa as relações entre os dados conhecidos (y e φ), segundo o princípio de mínimos quadrados, é aquele que minimiza a seguinte função de custo:

$$J(\theta, N) = \frac{1}{2}\sum_{k=1}^{N}(y(k) - \hat{y}(k))^2 \tag{4.2}$$

$$= \frac{1}{2}\sum_{k=1}^{N}[y(k) - (\theta_0 + \varphi_1(k)\theta_1 + ... + \varphi_n(k)\theta_n)]^2 \tag{4.3}$$

Se definirmos os seguintes vetores e matrizes:

$$Y = [y(1)y(2)...y(N)]^T \tag{4.4}$$

$$\theta = [\theta_0\theta_1...\theta_n]^T \tag{4.5}$$

$$\varphi = [1\varphi_1 \cdots \varphi_n]^T \tag{4.6}$$

$$\Phi = \begin{bmatrix} \varphi(1) \\ \varphi(2) \\ \vdots \\ \varphi(N) \end{bmatrix} \tag{4.7}$$

Podemos reescrever a equação 4.3 como:

$$J(\theta, N) = \frac{1}{2}(Y - \Phi\theta)^T(Y - \Phi\theta) \tag{4.8}$$

$$= \frac{1}{2}(Y^TY - Y^T\Phi\theta - \theta^T\Phi^TY + \theta^T\Phi^T\Phi\theta) \tag{4.9}$$

Como a matriz $\Phi^T\Phi$ é semi-definida positiva, ou seja nenhum dos valores característicos (*eigenvalues*) da matriz é negativo, a função $J(\theta, N)$ possui um mínimo [1]. Esse mínimo é localizado onde:

$$\frac{\partial J(\theta, N)}{\partial\theta} = 0 \tag{4.10}$$

$$\frac{1}{2}(-Y^T\Phi - Y^T\Phi + 2\theta^T\Phi^T\Phi) = 0 \tag{4.11}$$

$$-Y^T\Phi + \theta^T\Phi^T\Phi = 0 \tag{4.12}$$

$$\theta^T\Phi^T\Phi = Y^T\Phi \tag{4.13}$$

$$\Phi^T\Phi\theta = \Phi^TY \tag{4.14}$$

Assim, supondo que a matriz $\Phi^T\Phi$ é não singular, a solução para o problema dos mínimos quadrados é dada por:

$$\theta = (\Phi^T\Phi)^{-1}\Phi^TY \tag{4.15}$$

[1] Em analogia, recorde-se que $f(x) = ax^2 + bx + c$ tem máximo quando $a < 0$ e mínimo quando $a > 0$.

4.1.1.1 Identificação de sistemas dinâmicos pelo método de mínimos quadrados

Considere-se o sistema dinâmico descrito pela seguinte equação de diferenças:

$$y(k) + a_1 y(k-1) + ... + a_{n_a} y(k-n_a) = O + b_0 u(k-d_0) + ... + b_{n_b} u(k-d_0-n_b) + e(k) \quad (4.16)$$

Em que $e(k)$ é ruído branco, d_0 é o tempo morto discreto do sistema e O é um termo constante (*offset*). No caso particular de sistemas de controlo digitais, o modelo discreto possui sempre $d_0 \geq 1$, mesmo que o sistema contínuo a controlar possua um termo de proporcionalidade. Isto deve-se ao facto de primeiro se amostrar $y(k)$ e só depois se calcular $u(k)$. Havendo um termo de proporcionalidade no sistema contínuo, no momento de amostragem este é aplicado a $u(k-1)$ (para o caso particular de ser utilizado um extrapolador ZOH).

O modelo de regressão do sistema representado pela equação 4.16 é dado por:

$$y(k) = O - a_1 y(k-1) - ... - a_{n_a} y(k-n_a) + b_0 u(k-d_0) + ... + b_{n_b} u(k-d_0-n_b) \quad (4.17)$$

De acordo com a equação 4.1, os vetores de regressão φ e dos parâmetros desconhecidos θ são:

$$\varphi(k) = [1\ y(k-1)\ ...\ y(k-n)\ u(k-d_0)\ ...\ u(k-d_0-n_b)]^T \quad (4.18)$$

$$\theta = [O\ -a_1\ ...\ -a_{n_a}\ b_0\ ...\ b_{n_b}]^T \quad (4.19)$$

Definidos estes vetores pode-se estimar os parâmetros do sistema utilizando o método dos mínimos quadrados.

4.1.2 Algoritmos de identificação recursiva

Os algoritmos de identificação recursiva são essenciais em alguns esquemas de controlo adaptativo, nomeadamente as técnicas de adaptação contínua. A implementação recursiva permite que as estimativas de um ciclo anterior sejam utilizadas para o cálculo das estimativas atuais, economizando tempo e recursos computacionais.

Recordando as noções descritas na secção 3.3, a escolha do algoritmo de identificação é condicionada pela escolha do modelo de sistema. Por sua vez, a escolha adequada do modelo do sistema requer alguns conhecimentos sobre o sistema.

4.1.2.1 Método dos mínimos quadrados recursivo

Recorrer ao método de mínimos quadrados para estimar um sistema de cada vez que se obtém uma nova amostra resulta num grande desperdício de recursos computacionais (tempo de computação e memória). O método de mínimos quadrados recursivo resolve esse problema ao reformular as equações que compõe o método não-recursivo de tal forma que os resultados obtidos no instante de tempo $k-1$ possam ser utilizados no cálculo dos resultados no instante de tempo k. Esta reformulação permite poupar recursos computacionais sem sacrificar a qualidade dos resultados obtidos.

A dedução das equações que constituem o método de mínimos quadrados recursivo pode ser consultada em Mota [19]. As equações que constituem o método, para sistemas com apenas uma saída, são apresentadas em seguida:

$$\begin{cases} \epsilon(k) = y(k) - \varphi^T(k)\theta(k-1) \\ \\ K(k) = \frac{P(k-1)\varphi(k)}{1+\varphi^T(k)P(k-1)\varphi(k)} \\ \\ \theta(k) = \theta(k-1) + K(k)\epsilon(k) \\ \\ P(k) = \left(I - K(k)\varphi^T(k)\right) P(k-1) \end{cases} \tag{4.20}$$

Em que:

- $\epsilon(k)$ - é o vetor de erro de estimação
- $K(k)$ - é o vetor de ganho
- $P(k)$ - é a matriz de covariância

4.1.2.2 Método de Mínimos quadrados recursivo com esquecimento exponencial

Quando se pretende estimar e acompanhar os parâmetros de um sistema variante no tempo é necessário um algoritmo de identificação adequado. Isto deve-se ao facto de algoritmos como o usado no método de mínimos quadrados recursivo nunca eliminarem a influência de dados antigos. Ou seja, dados que não pertencem ao sistema atual são utilizados para estimar os seus parâmetros, degradando a qualidade da estimação.

O método de mínimos quadrados recursivo com esquecimento exponencial lida com este problema introduzindo uma constante de esquecimento λ ($0 < \lambda < 1$) que vai diminuindo progressivamente o peso dos dados mais antigos no cálculo dos parâmetros atuais do sistema.

Tal como para o método de mínimos quadrados recursivos, a dedução das equações deste método pode ser consultada em Mota [19]. As equações apresentadas de seguida são também apenas válidas para sistemas com uma saída.

$$\begin{cases} \epsilon(k) = y(k) - \varphi^T(k)\theta(k-1) \\ \\ K(k) = \frac{P(k-1)\varphi(k)}{\lambda+\varphi^T(k)P(k-1)\varphi(k)} \\ \\ \theta(k) = \theta(k-1) + K(k)\epsilon(k) \\ \\ P(k) = \frac{1}{\lambda}\left(I - K(k)\varphi^T(k)\right) P(k-1) \end{cases} \tag{4.21}$$

Normalmente, os valores utilizados para λ estão compreendidos entre 0.9 e 0.999 [19].

Este método, apesar de ser capaz de acompanhar a evolução do sistema observado possui uma desvantagem: apenas funciona bem havendo excitação constante. Quando o sistema não é excitado devidamente durante largos períodos de tempo, os valores de φ tendem para 0 e isto leva a que $P(k)$ se aproxime de $\frac{P(k-1)}{\lambda}$

$$
\begin{aligned}
P(k)|_{\varphi(k)\approx 0} &= \frac{1}{\lambda}\left(P(k-1) - \frac{P(k-1)\varphi(k)\varphi^T(k)P(k-1)}{\lambda + \varphi^T(k)P(k-1)\varphi(k)}\right)\Bigg|_{\varphi(k)\approx 0} \\
&\approx \frac{P(k-1)}{\lambda}
\end{aligned}
\tag{4.22}
$$

Como $\lambda < 1$, os valores da matriz P aumentam exponencialmente. Consequentemente, aumenta também a sensibilidade do estimador, levando a que mesmo quando $\varphi(k)$ possui valores pequenos sejam efetuadas correções excessivas aos parâmetros do sistema. Este fenómeno indesejável é conhecido como *estimator wind-up* [19].

4.1.2.3 Método de Mínimos Quadrados Recursivo com Esquecimento Direcional

A utilização de esquecimento direcional permite acompanhar a evolução dos parâmetros de um sistema variante no tempo sem que a matriz de covariância sofra *wind-up* quando há baixa excitação.

A ideia deste algoritmo é que dados antigos apenas sejam descartados quando existe excitação suficiente para produzir novos dados de interesse à estimação [20].

O método consegue isto ao aplicar o fator de esquecimento direcional α apenas aos dados que dão entrada no sistema, conforme é demonstrado por Mota [19]. As equações que descrevem o algoritmo são:

$$
\begin{cases}
\epsilon(k) = y(k) - \varphi^T(k)\theta(k-1) \\
\alpha(k) = \begin{cases} \lambda_{df} - \frac{1-\lambda_{df}}{\varphi^T(k)P(k-1)\varphi(k)} & se \;\; \varphi^T(k)P(k-1)\varphi(k) > 0 \\ 1 & se \;\; \varphi^T(k)P(k-1)\varphi(k) \approx 0 \end{cases} \\
K(k) = \frac{P(k-1)\varphi(k)}{1+\varphi^T(k)P(k-1)\varphi(k)\alpha(k)} \\
\theta(k) = \theta(k-1) + K(k)\epsilon(k) \\
P(k) = \left(I - K(k)\varphi^T(k)\alpha(k)\right)P(k-1)
\end{cases}
\tag{4.23}
$$

O parâmetro λ_{df} comporta-se como o parâmetro λ utilizado no método de esquecimento exponencial.

4.1.2.4 Método de Mínimos quadrados recursivo estendido

Quando as perturbações de um sistema não se comportam como ruído branco o modelo ARX usado para representar a dinâmica do sistema na equação 4.16 deixa de ser o adequado. Para estes casos deve-se recorrer a um modelo ARMAX:

$$
y(k) + \sum_{i=1}^{n_a} a_i y(k-i) = O + \sum_{i=0}^{n_b} b_i u(k-d_0-i) + e(k) + \sum_{i=1}^{n_c} c_i e(k-i) \tag{4.24}
$$

Para este tipo de sistemas os métodos de estimação apresentados não são adequados, pois $e(k)$ não é observável e como tal não pode ser utilizado como regressor. Conforme é demonstrado por Wellstead and Zarrop [21] a estimação dos sistemas através dos métodos apresentados leva a que, nestas condições, os parâmetros sejam estimados com erro. No entanto, com os algoritmos apresentados anteriormente é possível estimar um sistema com um comportamento dinâmico aproximado ao real aumentando o número de regressores [1].

Como alternativa, o método de mínimos quadrados recursivo estendido tem a vantagem de estimar separadamente os parâmetros da dinâmica associada às perturbações, permitindo obter um modelo do sistema com a ordem correta e cujos parâmetros são estimados com maior exatidão. Neste método considera-se que $e(k)$ pode ser aproximado por $\epsilon(k)$, sendo $\epsilon(k)$ usado como regressor [21].

Qualquer dos algoritmos apresentados neste capítulo é válido para o método de mínimos quadrados recursivo estendido. Apenas é necessário redefinir os vetores φ e θ, tal que :

$$\varphi(k) = [1\ y(k-1)\ ...\ y(k-n)\ u(k-d_0)\ ...\ u(k-d_0-n_b)\ \epsilon(k-1)\ ...\ \epsilon(k-n_c)]^T \quad (4.25)$$

$$\theta = [O\ -a_1\ ...\ -a_{n_a}\ b_0\ ...\ b_{n_b}\ c_1\ ...\ c_{n_c}]^T \quad (4.26)$$

4.2 Resolução da equação de Diophantine

A síntese do controlador por posicionamento de polos foi abordada na secção 3.2. Foram apresentadas as equações e as condições necessárias para o cálculo dos parâmetros do controlador. No entanto, os métodos para a resolução da equação de Diophantine não foram discutidos. Neste capítulo serão apresentados dois métodos de resolução para a equação.

Relembrar que a equação de Diophantine é dada por:

$$\begin{aligned} A_c &= AR_dR' + BS_dS' && (4.27)\\ &= A'R' + B'S' && (4.28) \end{aligned}$$

Notar, que para que a equação 4.28 tenha solução é necessário que os fatores comuns de A' e B' também sejam fatores de A_c [1].

4.2.1 Algoritmo de Kucera

Este é o método de resolução pela via polinomial. O algoritmo de Kucera, apresentado por Wellstead and Zarrop [21], permite determinar o maior divisor comum G dos polinómios A' e B', ou seja, permite determinar os fatores comuns dos dois polinómios. Permite também determinar a solução da equação 4.29, os polinómios X e Y, e a solução de ordem mínima para a equação 4.30, os polinómios U e V.

$$\begin{aligned} A'X + B'Y &= G && (4.29)\\ A'U + B'V &= 0 && (4.30) \end{aligned}$$

Estas equações podem ser expressas pelas seguintes matrizes:

$$\begin{bmatrix} X & Y \\ U & V \end{bmatrix} \begin{bmatrix} A' \\ B' \end{bmatrix} = \begin{bmatrix} G \\ 0 \end{bmatrix} \tag{4.31}$$

Isto implica que existe uma sequência de operações lineares que sendo aplicada às linhas da matriz $\begin{bmatrix} A' \\ B' \end{bmatrix}$ permite obter a matriz $\begin{bmatrix} G \\ 0 \end{bmatrix}$. Aplicando a mesma sequência de operações a uma matriz identidade obtemos $\begin{bmatrix} X & Y \\ U & V \end{bmatrix}$.

Para a resolução das equações 4.29 e 4.30 definam-se as matrizes M e N e siga-se o algoritmo indicado:

$$M = \begin{bmatrix} A' \\ B' \end{bmatrix} \quad N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.32}$$

1. Se um dos polinómios de M for igual a 0, ir para 4.
2. Defina-se β como o rácio entre os coeficientes do termo de maior grau do polinómio de maior grau de M e o termo de maior grau do outro polinómio. Defina-se Δ como a diferença entre os graus dos polinómios ($\Delta \geq 0$).
3. Subtraia-se ao polinómio de maior grau de M o produto do polinómio de grau menor e βz^{Δ}. Repita-se a mesma operação de linha na matriz N. Ir para 1.
4. Se o polinómio nulo estiver na primeira linha de M, trocar as linhas de M e N. Terminar.

Os resultados obtidos são:

$$M = \begin{bmatrix} G \\ 0 \end{bmatrix} \quad N = \begin{bmatrix} X & Y \\ U & V \end{bmatrix} \tag{4.33}$$

Uma solução particular para a equação 4.28 é dada por:

$$\begin{aligned} R'_0 &= X A_c \; div \; G \\ S'_0 &= Y A_c \; div \; G \end{aligned} \tag{4.34}$$

enquanto a solução geral é dada por:

$$\begin{aligned} R' &= R'_0 + QU \\ S' &= S'_0 - QV \end{aligned} \tag{4.35}$$

Conforme afirma Åström and Wittenmark [2], a solução de ordem mínima para a equação pode ser obtida definindo:

$$Q = -S'_0 \; div \; V \tag{4.36}$$

Por fim os polinómios R e S são dados por:

$$R = R' R_d \tag{4.37}$$

$$S = S' S_d \tag{4.38}$$

4.2.2 Matriz de Sylvester e eliminação Gaussiana

Recordando a secção 3.2, a solução de ordem mínima para a equação 4.28 implica que os polinómios envolvidos possuam os graus seguintes:

$$\begin{cases} deg(A') = deg(A) + deg(R_d) = n_a \\ deg(B') = deg(B) + deg(S_d) = n_b \\ deg(R') = deg(A) + deg(S_d) - 1 = n_r \\ deg(S') = deg(A) + deg(R_d) - 1 = n_s \\ deg(A_c) = na + nr = n_c \end{cases}$$

Considere-se que $deg(A) = deg(B)$ e que o atraso d inerente ao sistema (polos em excesso) seja representado pela atribuição de valor 0 ao primeiros d coeficientes do polinómio B. Considerem-se então os polinómios:

$$\begin{cases} A'(z^{-1}) = a'_0 + a'_1 z^{-1} + ... + a'_{n_a} z^{-n_a} \\ B'(z^{-1}) = b'_0 + b'_1 z^{-1} + ... + b'_{n_b} z^{-n_b} \\ R'(z^{-1}) = r'_0 + r'_1 z^{-1} + ... + r'_{n_r} z^{-n_r} \\ S'(z^{-1}) = s'_0 + s'_1 z^{-1} + ... + s'_{n_s} z^{-n_s} \\ A_c(z^{-1}) = c_0 + c_1 z^{-1} + ... + c_{n_c} z^{-n_c} \end{cases}$$

Podemos expressar a equação 4.28 como a operação matricial entre as matrizes seguintes:

$$\left[\begin{array}{cccc|cccc} a'_0 & 0 & \cdots & 0 & b'_0 & 0 & \cdots & 0 \\ a'_1 & a'_0 & \cdots & 0 & b'_1 & b'_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a'_{n_a-1} & a'_{n_a-2} & \cdots & a'_0 & b'_{n_b-1} & b'_{n_b-2} & \cdots & b'_0 \\ a'_{n_a} & a'_{n_a-1} & \cdots & a'_1 & b'_{n_b} & b'_{n_b-1} & \cdots & b'_1 \\ 0 & a'_{n_a} & \cdots & a'_2 & 0 & b'_{n_b} & \cdots & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a'_{n_a} & 0 & 0 & \cdots & b'_{n_b} \end{array}\right] \left[\begin{array}{c} r'_0 \\ \vdots \\ \vdots \\ r'_{n_r} \\ s'_0 \\ \vdots \\ \vdots \\ s'_{n_s} \end{array}\right] = \left[\begin{array}{c} c_0 \\ \vdots \\ \vdots \\ c_{n_r} \\ c_{n_r+1} \\ \vdots \\ \vdots \\ c_{n_c} \end{array}\right] \tag{4.39}$$

$\underbrace{\qquad\qquad}_{n_r+1}$ $\underbrace{\qquad\qquad}_{n_s+1}$

Quando temos uma situação do tipo $A \times B = C$ e A é uma matriz quadrada não-singular, então $A^{-1}[A|C] = [I|B]$. Isto significa que se pode obter $[I|B]$ a partir de $[A|C]$ através de

operações elementares entre as linhas. O conjunto dessas operações constitui o algoritmo de eliminação Gaussiana [22].

A matriz quadrada da esquerda na equação 4.39 é uma matriz de Sylvester. Este tipo de matrizes têm a propriedade de apenas serem não-singulares quando não existem fatores comuns entre A' e B'. Ou seja, por eliminação Gaussiana apenas se pode determinar os polinómios R' e S' na equação 4.39 se os polinómios A' e B' forem primos relativos.

Para resolver a equação de Diophantine por este método define-se uma matriz tal como a apresentada de seguida:

$$\begin{bmatrix} a'_0 & 0 & \cdots & 0 & b'_0 & 0 & \cdots & 0 & c_0 \\ a'_1 & a'_0 & \cdots & 0 & b'_1 & b'_0 & \cdots & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a'_{n_a-1} & a'_{n_a-2} & \cdots & a'_0 & b'_{n_b-1} & b'_{n_b-2} & \cdots & b'_0 & c_{n_r} \\ a'_{n_a} & a'_{n_a-1} & \cdots & a'_1 & b'_{n_b} & b'_{n_b-1} & \cdots & b'_1 & c_{n_r+1} \\ 0 & a'_{n_a} & \cdots & a'_2 & 0 & b'_{n_b} & \cdots & b'_2 & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{n_a} & 0 & 0 & \cdots & b'_{n_b} & c_{n_c} \end{bmatrix}$$

E aplicando o algoritmo de eliminação Gaussiana obtém-se a matriz:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & r'_0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & r'_{n_r} \\ 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & s'_0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & s'_{n_s} \end{bmatrix}$$

Os valores presentes na última coluna da matriz são então os coeficientes dos polinómios R' e S'.

Por fim os polinómios R e S são dados por:

$$R = R'R_d \tag{4.40}$$

$$S = S'S_d \tag{4.41}$$

4.3 Exemplos de aplicação em *MATLAB* e resultados

Dada a "naturalidade" com que em *MATLAB* se representam matrizes e como são realizadas operação matriciais, a implementação dos algoritmos de mínimos quadrados recursivo, mínimos quadrados recursivo com esquecimento exponencial e mínimos quadrados recursivo com esquecimento direcional, representados pelas equações 4.20, 4.21 e 4.23, foi direta e simples.

Por sua vez, os algoritmos de resolução da equação de Diophantine foram implementados de uma forma mais aproximada à que seria utilizada para uma implementação em *C*, não dependendo das capacidades do *MATLAB* para lidar com matrizes e tentando obter o máximo de eficiência.

Em particular, o método de resolução pela via polinomial é o que possui mais detalhes de implementação. Em *MATLAB* os polinómios são representados como matrizes $1 \times (n+1)$, sendo n a ordem do polinómio, preenchidas com os coeficientes dos polinómios. A primeira posição da matriz é ocupada pelo coeficiente do termo de maior grau e a última posição ocupada pelo coeficiente do termo de grau 0. Esta forma de representar polinómios como matrizes é extremamente ineficiente, obrigando a ajustes antes de realizar algumas operações ou após a realização das mesmas.

A título de exemplo considere-se o polinómio $A(x) = x^2+2x+3$ e o polinómio $B(x) = x+2$, representados em *MATLAB* por $A = [1, 2, 3]$ e $B = [1, 2]$. Para que seja possível somar estes dois polinómios em *MATLAB* é necessário adicionar um termo $0x^2$ ao polinómio B, de tal forma que $B = [0, 1, 2]$, antes de realizar a operação.

Se umas vezes é necessário adicionar 0's aos polinómios, em outras convém removê-los. Por exemplo, executando na linha de comandos do *MATLAB* $[Q, R] = deconv([1, 2, 3], [1, 1])$ são devolvidos os polinómios $Q = [1, 1]$ e $R = [0, 0, 2]$. O polinómio R é representado de uma forma pouco prática, para evitar situações destas é necessário verificar, nos resultados das operações, a existência de coeficientes nulos nos termos de maior ordem e definir novas matrizes nas quais estes termos não surjam.

Para a implementação do algoritmo de resolução de Diophantine pela via polinomial, e com vista a uma futura implementação em linguagem *C*, cada polinómio foi representado por um *array* de tamanho fixo $1 \times MAX$ e um inteiro que indica a ordem do polinómio. A forma como os coeficientes estão dispostos no *array* foi invertida, sendo a primeira posição ocupada pelo coeficiente do termo de ordem 0 e a última pelo coeficiente do termo de maior ordem. Os polinómios do exemplo da soma de polinómios seriam então dados, para um $MAX = 4$, pelo conjunto $A = [3, 2, 1, 0]$ e $nA = 2$ e o conjunto $B = [2, 1, 0, 0]$ e $nB = 1$. Com rotinas específicas para lidar com polinómios neste formato conseguem-se ganhos de performance. No caso do exemplo da deconvolução, para $MAX = 4$, $R = [2, 0, 0, 0]$ e seria apenas necessário verificar qual última posição de R ocupada por um valor diferente de 0 e ajustar o valor de nR, não sendo necessário declarar novas variáveis.

Os exemplos seguintes demonstram as capacidades dos algoritmos referidos nas secções 4.1 e 4.2, em diferentes cenários de ruído e comportamento dos parâmetros dos sistemas.

4.3.1 Mínimos quadrados na presença de ruído e parâmetros variáveis no tempo

Neste exemplo é utilizado o método de mínimos quadrados recursivo, sem qualquer fator de esquecimento, para estimar os parâmetros de dois sistemas, um invariante no tempo e outro variante no tempo. O objetivo é demonstrar as capacidades do algoritmo para estimar sistemas com ruído e sistema variantes no tempo.

O sistema invariante no tempo é representado pelo modelo de regressão:

$$y(k) = 1.5y(k-1) - 0.7y(k-2) + 1.2u(k) - 0.5u(k-2) + e(k)$$

Por sua vez, o sistema variante no tempo é representado por:

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) + 1.2u(k) - 0.5u(k-2) + e(k)$$

Em que para:

$$\begin{cases} k \leq 400 & \begin{cases} a_1 = -1.5 \\ a_2 = 0.7 \end{cases} \\ k > 400 & \begin{cases} a_1 = -1 \\ a_2 = 0.9 \end{cases} \end{cases}$$

Para qualquer um dos sistemas, $e(k)$ é um sinal do tipo ruído branco de média 0 e desvio padrão 0.05. O sinal $u(k)$ é um sinal do tipo *Pseudo-Random Binary Signal* (PRBS) de amplitude ± 1.

A simulação tem uma duração de 800 amostras e foram realizados 200 ensaios. Para cada um dos ensaios foi gerado um vetor de $e(k)$ e um vetor $u(k)$. A matriz de covariância foi inicializada como $10I$ e o vetor de coeficientes desconhecidos foi inicializado a 0's.

A figura seguinte apresenta a evolução das estimações dos parâmetros dos dois sistemas num dos ensaios. Em cima pode ver-se a estimação dos parâmetros do sistema invariante no tempo (parâmetros fixos) e em baixo a estimação dos parâmetros do sistema variante no tempo (parâmetros variáveis).

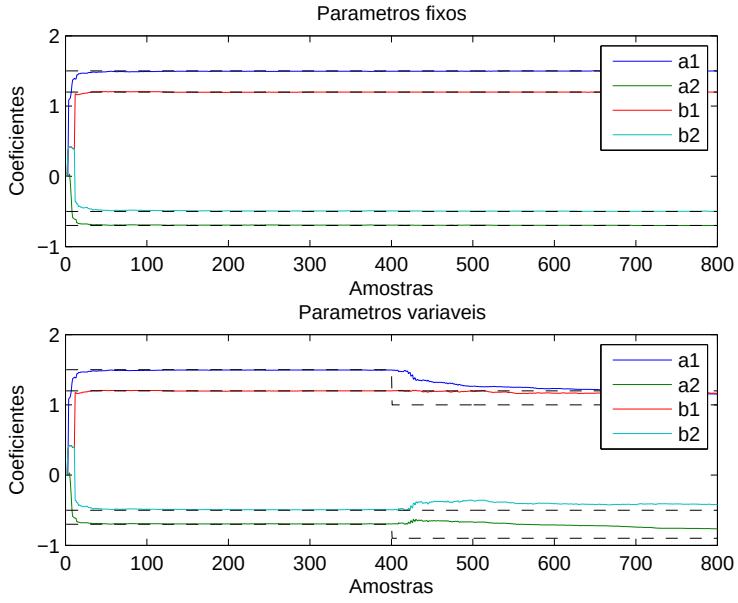


Figura 4.1: Estimação dos coeficientes de sistemas com parâmetros fixos (cima) e parâmetros variáveis (baixo) utilizando o método de mínimos quadrados recursivo

A tabela seguinte apresenta os valores de média e desvio-padrão das estimativas no momento $k = 800$, bem como os valores reais dos parâmetros nesse momento.

Coeficientes	Valor Real	Parâmetros Fixos		Parâmetros Variáveis	
		Média	Desvio-Padrão	Média	Desvio-Padrão
a_1	-1.5	-1.4993	0.0028	-1.1546	0.0026
a_2	0.7	0.6993	0.0024	0.7669	0.0020
b_1	1.2	1.1999	0.0016	1.1655	0.0025
b_2	-0.5	-0.4991	0.0037	-0.4224	0.0027

Tabela 4.1: Comparação das estimações de sistemas invariantes e variantes no tempo para $k = 800$

Como se pode constatar o método de mínimos quadrados é capaz de estimar corretamente os parâmetros de um sistema invariante no tempo mesmo na presença de ruído branco. No entanto, é incapaz de acompanhar de modo eficaz a evolução dos parâmetros de um sistema variante no tempo, isto deve-se ao facto de neste algoritmo nunca se descartarem dados antigos.

4.3.2 Comparação dos algoritmos de identificação de sistemas variantes no tempo.

Neste exemplo considere-se o sistema variante no tempo cujo o modelo de regressão é dado por:

$$y(k) = -a_1 y(k-1) - a_2(k-2) + 1.2u(k-1) - 0.5(k-2) + e(k)$$

Em que os coeficientes a_1 e a_2 dependem de k de acordo com:

$$\begin{cases} k \leq 1000 & \begin{cases} a_1 = -1.5 \\ a_2 = 0.7 \end{cases} \\ k > 1000 & \begin{cases} a_1 = -1 \\ a_2 = 0.9 \end{cases} \end{cases}$$

Mais uma vez, o sinal $e(k)$ é ruído branco de media 0 e desvio-padrão 0.05. O sinal $u(k)$ é do tipo PRBS com amplitude ± 1 durante as primeiras 500 amostras e é nulo nas restantes.

Cada uma das simulações teve uma duração de 1000 amostras e foram realizados 200 ensaios. Para as estimações foi utilizado o método de mínimos quadrados com esquecimento direcional e com esquecimento exponencial, em que $\lambda_{df} = \lambda = 0.9$. Para cada um dos ensaios foi gerado um vetor de $e(k)$ e um vetor $u(k)$. A matriz de covariância foi inicializada como $10I$ e o vetor de coeficientes desconhecidos foi inicializado a 0's.

A figura seguinte apresenta a evolução das estimações numa das repetições. Os gráficos de cima permitem comparar a evolução da estimação de parâmetros para os dois métodos, esquecimento direcional à esquerda e esquecimento exponencial à direita. Os gráficos de baixo permitem comparar o traço das matrizes de covariância ao longo do tempo, esquecimento direcional à esquerda e esquecimento exponencial à direita.

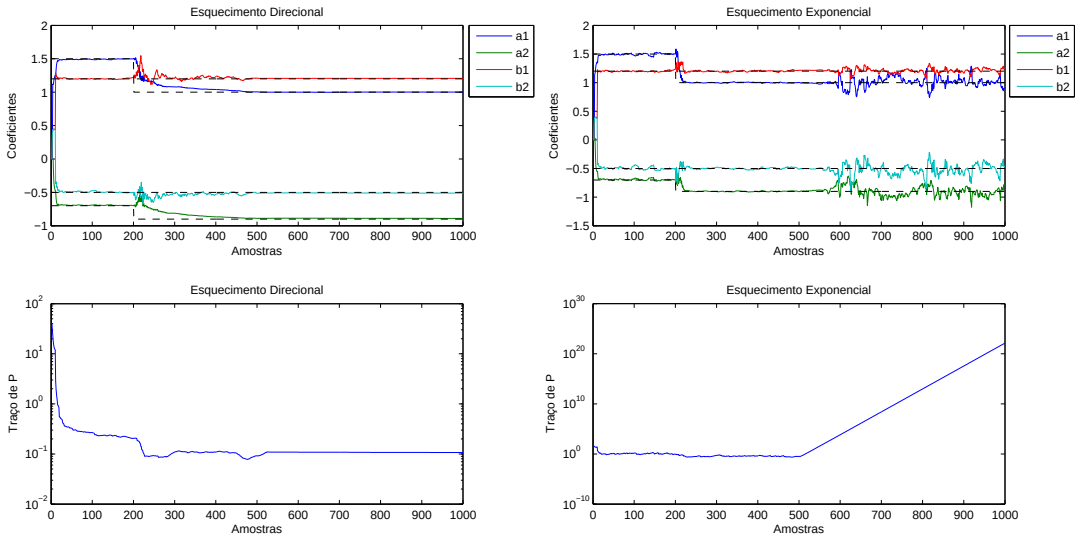


Figura 4.2: Estimação dos coeficientes de sistemas variantes no tempo (cima) e evolução do traço das matrizes de covariância (baixo).

Facilmente se conclui que o método de esquecimento exponencial é consideravelmente mais rápido a acompanhar a evolução dos parâmetros do sistema. No entanto, no gráfico do canto inferior direito é possível observar-se o *wind-up* da matriz de covariância quando o sistema deixa de ser excitado. Como resultado do *wind-up* da matriz de covariância, pode ver-se no canto superior direito a degradação das estimativas a partir de $k = 500$.

No canto inferior esquerdo pode observar-se que quando se utiliza esquecimento direcional, a pouca excitação do sistema para $k \geq 500$ não afeta o traço da matriz de covariância nem a qualidade das estimativas.

Na tabela seguinte pode comparar-se com mais detalhe a qualidade das estimativas para os dois métodos de esquecimento de fatores quando $k = 1000$.

Coeficientes	Valor Real	Esquecimento Direcional		Esquecimento Exponencial	
		Média	Desvio-Padrão	Média	Desvio-Padrão
a_1	1	1.0048	0.0026	0.9872	0.1096
a_2	-0.9	-0.8939	0.0024	-0.8580	0.1364
b_1	1.2	1.1968	0.0073	1.1947	0.0535
b_2	-0.5	-0.5140	0.0088	-0.5182	0.1313

Tabela 4.2: Comparação das estimações de sistemas variantes no tempo para $k = 1000$

Este exemplo permite-nos perceber as vantagens e desvantagens de cada um dos métodos apresentados.

O método de esquecimento exponencial deve ser evitado quando o sistema a estimar pode ser sujeito a longos períodos de baixa excitação. No entanto, é o método preferível quando é necessária uma maior rapidez na convergência das estimativas para os valores corretos.

Com o método de esquecimento direcional, havendo mudanças nos parâmetros do sistema a estimar, a convergência das estimativas para os valores corretos é extremamente lenta. A

favor este método tem o facto de poder ser usado para estimar os parâmetros de um sistema que poderá ser sujeito a longos períodos de baixa excitação.

4.3.3 Comparação dos algoritmos de identificação na presença de ruído colorido.

Considere-se o modelo de regressão seguinte:

$$y(k) = 1.5y(k-1) - 0.7y(k-2) + u(k-1) + 0.5u(k-2) + e(k) - e(k-1) - 0.8e(k-2)$$

Em que $e(k)$ é um sinal de ruído branco de média 0 e desvio-padrão 0.5 e $u(k)$ é um sinal do tipo PRBS e amplitude ± 1.

Cada simulação decorreu durante 3000 amostras e foram executados 200 ensaios. Para cada um dos ensaios foi gerado um vetor de $e(k)$ e um vetor $u(k)$. A matriz de covariância foi inicializada como $10I$ e o vetor de coeficientes desconhecidos foi inicializado a 0's.

Este exemplo procura comparar as duas alternativas referidas neste capítulo para a estimação de um sistema dinâmico sujeito a ruído colorido, aumentar o número de regressores ou utilizar o método de mínimos quadrados regressivo estendido. O método de mínimos quadrados recursivo foi aplicado com 4, 6 e 8 regressores para estimar, respetivamente, modelos de 2ª, 3ª e 4ª ordem. O método de mínimos quadrados recursivo estendido foi aplicado com um total de 6 regressores para estimar um modelo de 2ª ordem.

A figura seguinte apresenta a estimação dos parâmetros dos diferentes modelos. A sigla RELS, de *Recursive Extended Least Squares,* refere-se ao método de mínimos quadrados recursivo estendido. A sigla RLS, de *Recursive Least Squares*, refere-se ao método de mínimos quadrados recursivo.

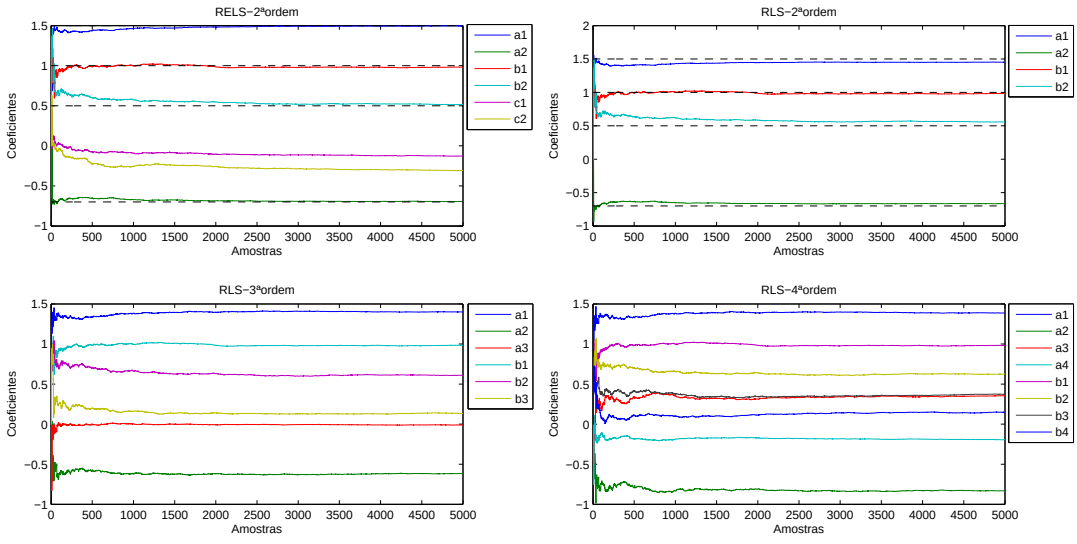


Figura 4.3: Evolução das estimações dos coeficientes dos sistemas.

Na figura do canto superior direito, os parâmetros do sistema de 2ª ordem estimados com o método de mínimos quadrados recursivo tendem para valores errados, tal como foi

referido na subsecção 4.1.2.4. Na figura do canto superior esquerdo, apesar dos coeficientes dos polinómios A e B convergirem para os valores corretos, os coeficientes do polinómio C convergem para valores errados (a identificação dos polinómios está de acordo com os modelos apresentados na secção 3.3).

A comparação das duas estimativas de sistemas de 2ª ordem pode ser feita com mais detalhe com base nos dados da tabela seguinte. Os valores registados resultam da análise dos dados recolhidos nos vários ensaios quando $k = 3000$.

Coeficiente	Valor Real	RELS 2ª ordem		RLS 2ª ordem	
		Média	Desvio-Padrão	Média	Desvio-Padrão
a_1	1.5	1.4895	0.0170	1.4531	0.0072
a_2	-0.7	-0.6927	0.0130	-0.6675	0.0069
b_1	1	1.0000	0.0135	0.9985	0.0142
b_2	0.5	0.5101	0.0242	0.5466	0.0168
c_1	-1	-0.0928	0.0486	-	-
c_2	-0.8	-0.2659	0.1063	-	-

Tabela 4.3: Comparação das estimações dos coeficientes dos dois modelos de 2ª ordem

Confirma-se que o método RELS permite obter boas estimativas dos coeficientes dos polinómios A e B. No entanto, os coeficientes do polinómio C não convergem para os valores corretos. Algoritmos de estimação como os que utilizam o critério de máxima verosimilhança poderão estimar com maior exatidão estes coeficientes [1].

A figura seguinte apresenta a resposta a degrau de cada um dos sistemas estimados, os valores dos coeficientes utilizados são os valores médios calculados a partir dos dados recolhidos nos vários ensaios.

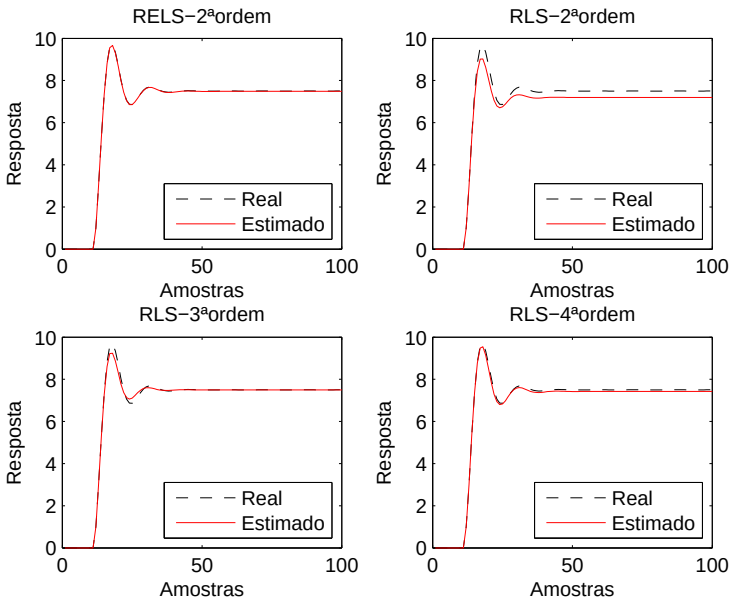


Figura 4.4: Resposta a degrau dos vários modelos estimados.

Analisando a figura rapidamente se conclui que o modelo de 2ª ordem estimado pelo método de mínimos quadrados recursivo estendido é apenas equiparável, em termos de comportamento, ao sistema de 4ª ordem estimado pelo método de mínimos quadrados recursivo.

4.3.4 Comparação dos algoritmos para a resolução da equação de Diophantine.

A comparação de performance foi realizada na seguinte plataforma:

Processador	Intel i5-2410M
Memória	4 GB
Sistema Operativo	Windows 7 64-bits
Software	MATLAB 7.12.0 (R2011a)

O teste consistiu na resolução da equação de Diophantine com os seguintes polinómios conhecidos:

$$\begin{aligned} A(z) &= z^2 - 1.724z + 0.741 \\ B(z) &= 0.009z + 0.007 \\ A_c(z) &= z^3 - 1.5z^2 + 0.7z \end{aligned}$$

Ambos os algoritmos detetaram a solução de ordem mínima correta:

$$\begin{aligned} R(z) &= z + 0.0632 \\ S(z) &= 17.9664z - 6.5951 \end{aligned}$$

Os algoritmos foram executados 50000 vezes e foram registados os valores de latência de computação. A partir destas medições determinou-se o tempo médio de execução e realizou-se uma análise comparativa aos dois algoritmos. Os resultados obtidos são apresentados na figura e tabela seguinte.

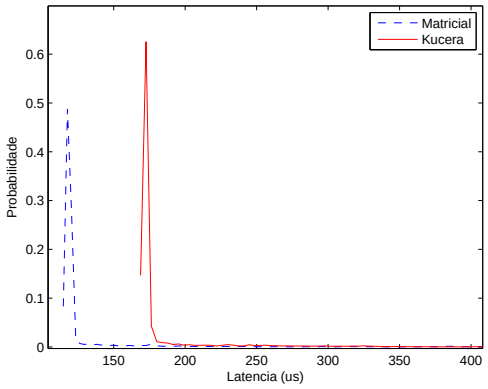


Figura 4.5: Distribuição de probabilidade para a latência dos algoritmos de resolução da equação de Diophantine

	Média (μs)
Matricial	127
Kucera	183

Tabela 4.4: Comparação da latência de computação dos algoritmos de resolução da equação de Diophantine

As conclusões a retirar desta experiência são: a via matricial é a mais rápida das duas a resolver a equação de Diophantine e a via polinomial é a mais robusta das duas.

A robustez da via polinomial deve-se a ser a única das duas vias capaz de resolver equações de Diophantine com fatores comuns entres os polinómios conhecidos. A via matricial não é capaz de resolver equações de Diophantine nas mesmas condições, pois uma matriz de Sylvester tem a propriedade de apenas ter inversa quando não existem fatores comuns entre os polinómios que representa.

O código de *Matlab* utilizado para esta simulação pode ser consultado no apêndice D.

4.4 Simulação em *Xenomai Lab*

Para validar os algoritmos desenvolvidos e a sua implementação em *Xenomai Lab*, realizaram-se simulações de um sistema de controlo adaptativo em *MATLAB* e *Xenomai Lab* e compararam-se os resultados.

Para a implementação em *Xenomai Lab* foi desenvolvida uma biblioteca de manipulação de matrizes, denominada *mtrx*, com o objetivo de simplificar a implementação dos algoritmos de identificação de sistemas e minimizar a probabilidade de erros de código. Esta biblioteca permite aproximar a sintaxe utilizada para operações matriciais em *C* à sintaxe utilizada no *MATLAB*. Uma descrição detalhada da biblioteca pode ser consultada no apêndice A.

O algoritmo de identificação de sistemas escolhido foi o algoritmo de mínimos quadrados recursivo com esquecimento direcional, em que $\lambda_{df} = 0.98$. Para a resolução da equação de Diophantine optou-se pela via matricial, apesar das desvantagens em relação ao método polinomial, de forma a tirar partido da simplicidade de utilização da biblioteca *mtrx*. E por fim, o controlador é projetado seguindo as condições apresentadas na secção 3.2.

Os polinómios que caracterizam o processo a controlar e os polinómios que caracterizam a dinâmica do sistema desejado são:

$$\begin{aligned} A(z) &= z^2 - 1.6065z + 0.6065 \\ B(z) &= 0.1065z + 0.0902 \\ A_m(z) &= z^2 - 1.3205z + 0.4966 \\ A_o(z) &= z \end{aligned}$$

Os parâmetros do controlador foram obtidos corretamente nas duas simulações, sendo caracterizado pelos polinómios seguintes:

$$\begin{aligned} R(z) &= z + 0.1104 \\ S(z) &= 1.6374z - 0.7445 \\ T(z) &= 0.8929z \end{aligned}$$

O sistema em malha fechada é caracterizado pelos polinómios:

$$\begin{aligned} A_m &= z^2 - 1.3205z + 0.4966 \\ B_m &= 0.0951z + 0.0805 \end{aligned}$$

Tanto o sistema $\frac{B}{A}$ como o sistema $\frac{B_m}{A_m}$ são sistemas estáveis. A frequência de amostragem definida para o sistema em *Xenomai Lab*, apesar de irrelevante pois o sistema é apenas uma simulação no domínio discreto, é de 10Hz.

Na figura seguinte pode ver-se o comportamento do sistema de controlo em *Xenomai Lab* à esquerda. O sinal de referência é apresentado o preto, o sinal de controlo a azul e o sinal da saída do sistema a vermelho. À direita apresenta-se o conjunto de blocos que compõe o sistema de controlo adaptativo em *Xenomai Lab*.

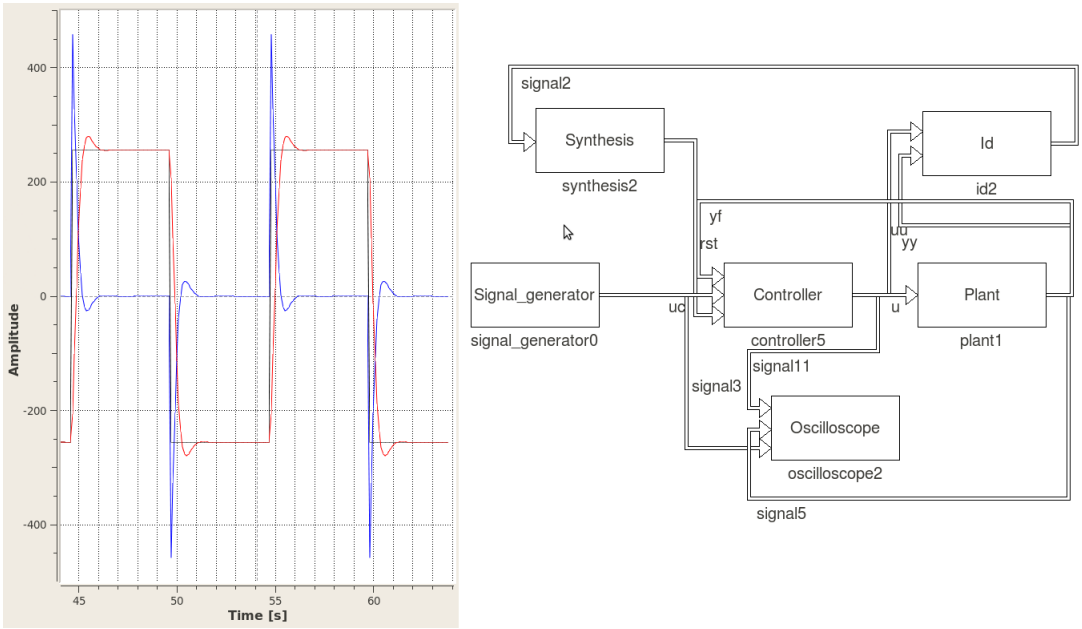


Figura 4.6: Sinais do sistema de controlo adaptativo em *Xenomai Lab* (esquerda) e blocos que o constituem (direita)

A figura seguinte apresenta a mesma simulação, mas desta vez executada em *MATLAB*.

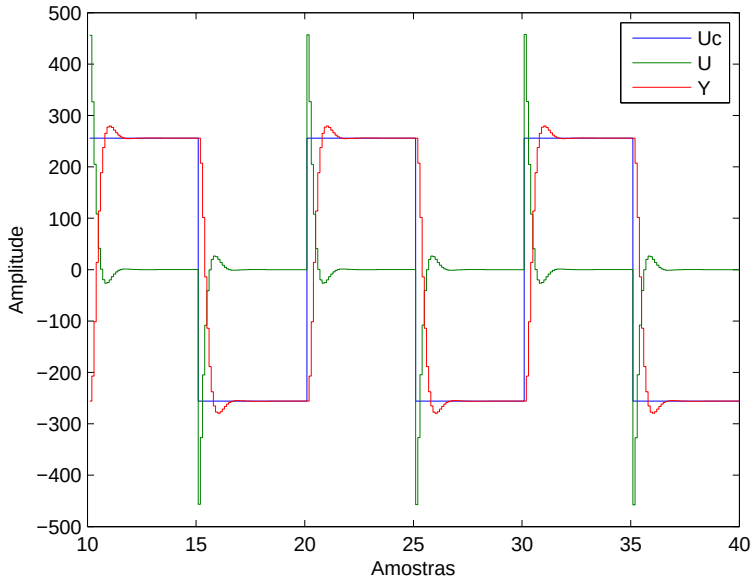


Figura 4.7: Sinais do sistema de controlo adaptativo em *MATLAB*

Comparando os resultados obtidos na simulação em *Xenomai Lab* com a simulação em *MATLAB*, pode considerar-se que a implementação dos algoritmos em *Xenomai Lab* foi efetuada corretamente, pois os sinais e os polinómios calculados são iguais nas duas simulações.

No entanto, no decorrer do trabalho foi detetada uma limitação funcional no *Xenomai Lab*. Quando existem malhas de realimentação no diagrama de blocos, o comportamento de alguns blocos e os próprios nomes atribuídos aos sinais podem levar a que o escalonamento dos processos não respeite o seu fluxo logicamente correto, introduzindo atrasos indesejados em sinais. No caso do esquema apresentado na figura 4.6, felizmente, o atraso de uma amostra foi aplicado ao sinal de entrada do bloco *Plant*. Isto é facilmente compensado se virmos o bloco *Plant* como possuindo um atraso de uma amostra implícito, bastando definir o polinómio B como $0.1065z^2 + 0.0902z$ para que o sistema se comporte como é suposto.

Por este motivo, a implementação final em *Xenomai Lab* do controlador adaptativo será, no âmbito deste trabalho, realizada com um único bloco.

Capítulo 5

Interface A-D e D-A

Num mundo de sistemas a controlar predominantemente analógicos, os controladores analógicos eram até há umas décadas a única opção disponível para alguém que procurasse controlar um processo. Hoje em dia, não só se utilizam controladores digitais, como estes, praticamente, se tornaram os controladores de eleição.

No entanto, dois sistemas de natureza distinta, como é o caso dos sistemas analógicos e os sistemas digitais, não podem interagir diretamente. Existe a necessidade de uma interface entre eles, capaz de converter os sinais entre ambos os domínios.

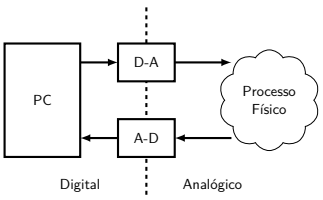


Figura 5.1: Interface entre domínios diferentes

Uma interface apropriada para um sistema analógico e um digital é composta por conversores analógicos-digitais (A-D) e conversores digitais-analógicos (D-A) e pelos transdutores específicos para a aplicação.

Como, no âmbito desta dissertação, o sistema a controlar é analógico e o controlador é um algoritmo executado num PC, logo um sistema digital, é necessário uma interface. Essa mesma interface pode ser implementada de diferentes maneiras. No entanto das pesquisas realizadas[23, 24, 25, 26] destacam-se três alternativas: placa de aquisição de dados, o uso de uma placa de áudio ou o desenvolvimento de uma placa baseada num micro-controlador. Cada uma das alternativas possui as suas vantagens e desvantagens que foram analisadas cuidadosamente de forma a determinar qual a alternativa mais adequada para a implementação da interface.

As placas de aquisição de dados oferecem um bom desempenho e um elevado grau de fiabilidade nas suas medições, no entanto têm preços elevados. A título de exemplo olhemos para uma das placas de baixa gama da empresa *National Instruments* que suporta tanto entradas como saídas analógicas, a PCI-6010. Esta placa tem um preço de 399€ pela placa PCI apenas [27]. Os cabos e conectores necessários são pagos à parte e aumentam consideravelmente os custos do pacote. Existe também a questão do suporte para este tipo de placas por parte do

Xenomai, o que limita ainda mais a lista de opções disponíveis no mercado. Assim, e dado o orçamento limitado para esta dissertação, foi posta de parte a opção da compra de uma placa de aquisição de dados.

O uso da placa de áudio do PC como interface é vantajosa se tivermos em conta que praticamente todos os PC's vêm equipados com uma, e que esta já contém tanto conversores A-D como D-A. No entanto é uma opção que sai penalizada quer pela programação complexa que requer, quer por apenas operar na gama de frequências audíveis para humanos. O último defeito referido é o que inviabiliza esta hipótese pois ter uma interface capaz de lidar com a componente DC de um sinal é essencial para controlo.

A terceira opção, e a que acabou por ser escolhida, é o desenvolvimento de uma placa simples, baseada num micro-controlador, para a interface. A principal justificação para esta decisão é a versatilidade dos micro-controladores atuais. A grande maioria vem equipada com uma gama alargada de módulos de comunicação, múltiplas entradas analógicas para o conversor A-D e possuem também uma capacidade de processamento elevada. A existência no DETI de uma placa de desenvolvimento baseada na família PIC32 da *Microchip* também pesou bastante na decisão, visto que acelera consideravelmente o processo de desenvolvimento da interface. Esta placa de desenvolvimento, de nome DETPIC32 [28], vem equipada com um micro-controlador PIC32MX795F512H e um conversor UART-USB de forma a facilitar a comunicação com o PC.

O desenvolvimento da interface não pode começar sem que se definam alguns requisitos para a placa. Os requisitos a definir referem-se à amplitude dos sinais analógicos com que a interface consegue lidar, ao número de entradas e saídas analógicas e aos métodos de comunicação entre a interface e o PC. Assim, definiu-se que a interface terá de ser capaz de lidar com sinais de amplitude compreendida entre -5 e 5 Volts, terá de suportar 4 entradas e 2 saídas analógicas e terá de ser capaz de comunicar com um PC através de uma porta série virtual (USB) e através de uma porta paralela com barramento de dados de 8 *bits* (*Centronics*).

Este capítulo apresenta o desenvolvimento da placa de interface A-D e D-A. Começando pela estrutura funcional da interface, segue-se uma explicação sobre os métodos de comunicação entre o PC e a interface e a finalizar são também apresentados resultados de medições realizadas com o intuito de avaliar a desempenho da interface.

De notar que o *hardware* apresentado possui apenas 1 entrada e 1 saída analógicas. Isto deve-se ao facto de ter-se construído apenas um protótipo, para depuração e validação. No entanto toda a infraestrutura de comunicações e software está pronta para as 4 entradas e 2 saídas, faltando apenas os circuitos de acondicionamento de sinal das mesmas.

5.1 Estrutura

A figura 5.2 apresenta a decomposição em blocos da interface.

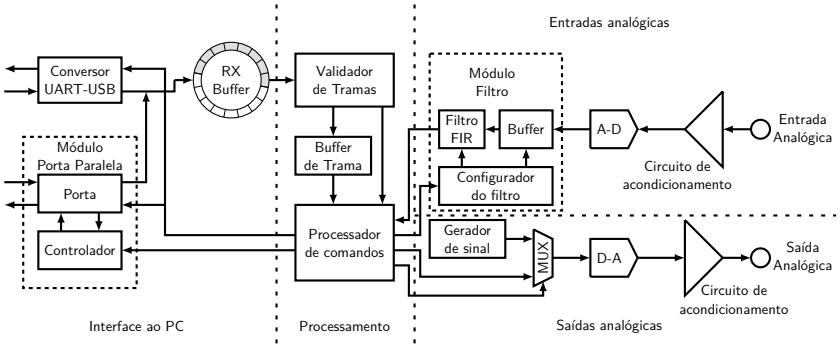


Figura 5.2: Esquemático da interface

Os blocos da interface podem ser agrupados em quatro conjuntos, cada um responsável por um conjunto de tarefas relacionadas. Esses conjuntos são o *Interface ao PC*, o *Processamento*, o *Entradas analógicas* e *o Saídas analógicas*.

O conjunto *Interface ao PC* é responsável pela comunicação entre o PC e a placa de interface. É composto pelos blocos *Conversor UART-USB*, pelo *Módulo Porta Paralela* e pelo *buffer* de receção *RX Buffer*. O bloco *Módulo Porta Paralela* pode ser decomposto em outros dois blocos mais simples a *Porta* propriamente dita e o *Controlador* do módulo.

O conjunto *Processamento* descodifica os comandos recebidos e coordena a placa para que esses comandos sejam executados. É composto pelos blocos *Validador de trama*, *Buffer de trama* e *Processador de comandos*.

Para a amostragem das entradas analógicas da interface há um outro conjunto de blocos, o conjunto *Entradas analógicas*. O *Circuito de acondicionamento* de sinal, o conversor *A-D* e o *Módulo Filtro*, este sendo constituído por *Filtro FIR*, *Buffer* e *Configurador de filtro*.

A finalizar, o conjunto responsável pela atuação sobre as saídas analógicas é o *Saídas analógicas*. Os blocos que o constituem são o *Gerador de sinal*, o *Mux*, o conversor *D-A* e o seu *Circuito de acondicionamento*.

5.1.1 Interface ao PC

No conjunto *Interface ao PC* o *Conversor UART-USB* e o *Módulo Porta Paralela* servem o mesmo propósito. É por onde os dados trocados entre o PC e a interface passam. Os dados oriundos do PC seguem para o *RX Buffer* e os dados que têm como destinatário o PC são provenientes do *Processador de comandos*.

A ligação entre o *Controlador* do *Módulo Porta Paralela* e o *Processador de comandos* serve para que o último inicie negociações necessárias à troca de dados entre a interface e o PC através da porta paralela.

5.1.2 Processamento

Neste conjunto o bloco *Validador de Tramas* serve para reconstruir as tramas a partir dos bytes recebidos e armazenados em *RX Buffer*. No caso de conseguir reconstruir uma trama válida este bloco armazena-a no *Buffer de Trama* e avisa o *Processador de comandos*. Caso contrário os bytes da trama inválida são descartados. Uma trama pode ser considerada inválida por estar incompleta ou pela deteção de corrupção dos seus dados através de algoritmos de verificação de erros.

O bloco *Processador de comandos* é o núcleo da interface. É o bloco responsável pela descodificação das tramas e execução dos comandos recebidos, coordenando os blocos da interface para esse efeito.

5.1.3 Entradas analógicas

O *Circuito de acondicionamento* de sinal destina-se a converter o sinal de entrada, que pode variar entre -5V e +5V, num sinal dentro da gama de entrada do conversor *A-D*, atenuando o sinal e adicionando o *offset* necessário.

O conversor *A-D* amostra periodicamente as entradas analógicas, guardando as amostras no *Buffer* do *Módulo Filtro*. A interface usa semântica de estado, mantendo uma imagem atualizada do sistema físico ao contrário de fazer uma leitura quando requisitada, com o objetivo de minimizar a duração total do processo de requisição de uma amostra pelo PC.

As várias amostras armazenadas no *Buffer* são filtradas por *Filtro FIR*, um filtro de média, apenas quando o PC requisita uma amostra e o resultado dessa filtragem é enviado logo de seguida através do *Processador de comandos*. A dimensão de *Buffer* e o os parâmetros do *Filtro FIR* podem ser ajustados pelo utilizador através do *Configurador do filtro*, havendo uma trama específica para esse efeito.

5.1.4 Saídas analógicas

Neste conjunto é no *Mux* que se seleciona o que se pretende para a saída analógica, sendo essa escolha efetuada entre o sinal de teste gerado pelo *Gerador de sinal* e o valor definido pelo utilizador no PC, este valor é passado ao *Mux* pelo *Processador de comandos*.

À saída do conversor *D-A* temos o *Circuito de acondicionamento*, este circuito amplifica o sinal do conversor *D-A* e adiciona-lhe o *offset* necessário para que à saída analógica o sinal respeite a gama definida de -5V a +5V.

5.2 Comunicação

Fisicamente é possível estabelecer a ligação entre o PC e a interface de duas formas diferentes, uma sendo por USB e outra através de porta paralela.

As diferentes formas de estabelecer uma ligação física entre o PC e a interface foram estabelecidas como forma de responder aos requisitos de diferentes tipos de utilizadores.

Assim se um utilizador procurar um tipo de ligação mais simples e estiver disposto a sacrificar alguma performance em troca dessa simplicidade, esse utilizador deverá utilizar a ligação por USB.

No entanto, este tipo de ligação será de evitar para um utilizador que procure trabalhar com frequências de amostragem elevadas visto que a frequência dos ciclos de uma ligação USB é de 1KHz [29, 30]. Outro motivo para evitar a ligação por USB por parte de alguns utilizadores é a inexistência de *drivers* com comportamento determinístico para sistemas operativos de tempo-real. Este último motivo apontado é de particular interesse para o trabalho realizado no âmbito desta dissertação, dado que o *software Xenomai Lab* corre sobre *Xenomai*, um sistema operativo de tempo-real.

A ligação por porta paralela é portanto a ligação adequada a utilizadores interessados em trabalhar com frequências de amostragem mais elevadas e/ou precisarem de trabalhar em sistemas operativos de tempo-real, visto que estes possuem *drivers* adequados para o módulo

de *hardware* da porta paralela. Assim, para o trabalho realizado nesta dissertação, a ligação por porta paralela é ligação de eleição.

5.2.1 Protocolo

Dado que a comunicação entre o PC e a interface precisa acomodar diversos tipos de comandos diferentes, cada um deles com características próprias, e que a direção do fluxo de tramas poderá ser invertida no decorrer da comunicação é necessário estabelecer um protocolo.

Mais especificamente, é necessário que o protocolo seja capaz de endereçar as 2 saídas e 4 entradas analógicas da interface e de acomodar o grupo de 5 comandos indicados na tabela 5.1.

O protocolo também prevê o uso de um algoritmo de verificação de erros de forma a acautelar situações em que a trama seja corrompida durante a transmissão.

No protocolo estabelecido uma trama genérica é um conjunto de *bytes* delimitados pelos *bytes SOT* (*Start Of Transmission*) e *EOT* (*End Of Transmission*) (figura 5.3). Ambos os *bytes* são de valor fixo e conhecido pelos dois lados da comunicação, servem para que quer o PC quer a interface sejam capazes de identificar a trama entre todos os bytes recebidos. O *byte* que inicia uma trama é o *byte SOT* e o *byte EOT* é portanto o seu terminador.

O segundo *byte*, denominado de *CMD*, codifica o comando a executar bem como o(s) endereço(s) da(s) entrada(s)/saída(s) analógica(s). Os quatro *bits* mais significativos são os *bits* reservados para codificar os comandos e os quatro menos significativos são os *bits* de endereçamento. Cada um dos bits de endereçamento identifica uma entrada/saída analógica permitindo assim que múltiplas entradas/saídas sejam endereçadas de uma única vez. Usa-se apenas um *byte* de forma a diminuir o *overhead* de comunicação e também por ser o suficiente para cumprir todos os requisitos do projeto, ou seja, acomodar os cinco comandos e os 4 endereços possíveis.

Segue-se o campo de dados, um conjunto de *bytes* de número variável e dependente do tipo de tramas. Por norma este campo serve para transmitir valores codificados em ASCII entre o PC e a interface. No entanto preveem-se situações em que não hajam quaisquer *bytes* deste tipo numa trama.

Por fim, de forma a acautelar a possibilidade de ocorrência de erros de transmissão, adiciona-se o par de *bytes* CRC_0 e CRC_1. Estes dois *bytes* transportam um valor calculado pelo emissor usando o algoritmo de verificação cíclica redundante do protocolo *Simple Serial Protocol* [31]. O recetor usa o mesmo algoritmo para calcular um valor a partir dos *bytes CMD* e *DATA*. De seguida compara o valor calculado com o valor registado nos *bytes CRC* e, em caso de igualdade, a mensagem é considerada como estando íntegra.

$$[SOT]\,[CMD]\,[DATA_0]\,...\,[DATA_n]\,[CRC_0][CRC_1]\,[EOT]$$

Figura 5.3: Trama genérica

Sempre que uma trama recebida é validada, o recetor procede ao seu processamento. Este protocolo estabelece uma relação de *Master-Slave* entre o PC e a interface, sendo o PC o *Master*, e onde todos os comandos fluem do PC para a interface havendo apenas respostas em casos específicos.

A não existência de tramas de *Acknowledgement* foi uma decisão de projeto com o intuito de diminuir o tempo despendido em comunicações. Esta decisão é suportada pela fiabilidade da interface e das ligações estabelecidas.

Na tabela 5.1 estão apresentados os 5 comandos definidos, juntos com o seu código e uma pequena descrição da sua função.

Os comandos *TEST* (figura 5.4) e *RESET* (figura 5.5) são os mais simples. São comandos que não possuem resposta nem qualquer *byte* de dados. Assim que o PC os transmite só resta à interface recebe-los e proceder à tarefa correspondente.

$$[SOT]\,[TEST]\,[CRC_0][CRC_1]\,[EOT]$$

Figura 5.4: Trama de *TEST*

$$[SOT]\,[RESET]\,[CRC_0][CRC_1]\,[EOT]$$

Figura 5.5: Trama de *RESET*

Por sua vez, os sinais *WRITE* e *FILTER* não possuem qualquer resposta mas possuem *bytes* de dados. O comando *WRITE* (figura 5.6) possui quatro *bytes* de dados por cada saída endereçada e o *FILTER* (figura 5.7) possui quatro *byte* de dados. É necessário especial atenção ao valor enviado pelo comando *FILTER* pois o filtro é dimensionado em potências de 2 tendo uma dimensão máxima de 64 amostras. Logo, para serem válidos os valores têm de números inteiros no intervalo fechado entre 0 e 6.

$$[SOT]\,[WRITE]\,[DATA_0]\,...\,[DATA_n]\,[CRC_0][CRC_1]\,[EOT]$$

Figura 5.6: Trama de *WRITE*

$$[SOT]\,[FILTER]\,[DATA_0]\,...\,[DATA_3]\,[CRC_0][CRC_1]\,[EOT]$$

Figura 5.7: Trama de *FILTER*

O comando *READ* implica uma resposta, pelo que se processa em duas fases. Numa primeira fase o PC envia uma trama, sem qualquer *byte* de dados, para interface, indicando apenas no *byte CMD* que está interessado em leituras de todas as portas analógicas endereçadas (figura 5.8). Na segunda fase a interface responde com uma trama composta pelo mesmo byte *CMD* e com quatro *bytes* de dados por cada entrada endereçada, nos quais são enviadas as leituras das entradas codificadas em ASCII (figura 5.9).

$$[SOT]\,[READ]\,[CRC_0][CRC_1]\,[EOT]$$

Figura 5.8: Tramas de *READ* (comando)

$$[SOT]\,[READ]\,[DATA_0]\,...\,[DATA_n]\,[CRC_0][CRC_1]\,[EOT]$$

Figura 5.9: Tramas de *READ* (resposta)

Comando	*bits*	Descrição
TEST	0x00	É colocado um sinal de teste nas saídas analógicas.
READ	0x10	São requisitadas leituras das entradas analógicas endereçadas
WRITE	0x20	Os valores indicados são escritos nas saídas analógicas endereçadas
RESET	0x30	As saídas analógicas são colocadas as 0 Volts
FILTER	0x40	Configura os filtros das entradas analógicas

Tabela 5.1: Comandos do protocolo da interface

5.2.2 Camada física

De forma a permitir o suporte das ligações por USB e porta paralela por parte da interface foi necessário implementar alguns módulos compostos por *software* e *hardware*.

A ligação por USB vem já implementada como parte integrante da placa DETPIC32, sendo suportada por um conversor UART-USB da empresa *FTDI Chip* [32]. Com o uso de *drivers* adequados [33] no PC esta ligação poderá ser encarada como uma ligação UART ponto-a-ponto o que simplifica de forma considerável o trabalho do utilizador.

A ligação de UART no micro-controlador foi configurada com um *baudrate* de 115200 símbolos por segundo, 8 *bits* de dados, 1 *stop bit* e sem qualquer *handshake*. Estas são as configurações que deverão ser usadas no PC com os *drivers Virtual COM Port* (VCP) da *FTDI Chip*.

Por sua vez, a ligação por porta paralela, apesar de ser baseada no *Byte Mode* do protocolo IEEE 1284 [34], tem sinais de controlo próprios e como tal não respeita nenhum dos modos de operação definidos para o *hardware*.

A ligação por porta paralela requer que se construa uma porta paralela do lado da interface. Assim, e tirando partido de certas características do micro-controlador tais como pinos tolerantes a 5 Volts e a possibilidade de os configurar como *open-drain*, construiu-se uma porta paralela do lado do micro-controlador capaz de respeitar os níveis lógicos TTL usados pelo *hardware* do lado do PC apenas com resistências de *pull-up* (apêndice C). Essa porta paralela é composta por 8 pinos de um porto usados para acomodar o barramento de dados e por 4 pinos suplementares usados para acomodar os sinais de controlo. Um destes pinos suporta a geração de interrupções a partir de uma fonte externa e é usado para assinalar a presença de um novo *byte* transmitido pelo PC no barramento de dados. A figura 5.10 apresenta em mais detalhe, no que refere à pinagem, a ligação entre o PC e o micro-controlador na interface.

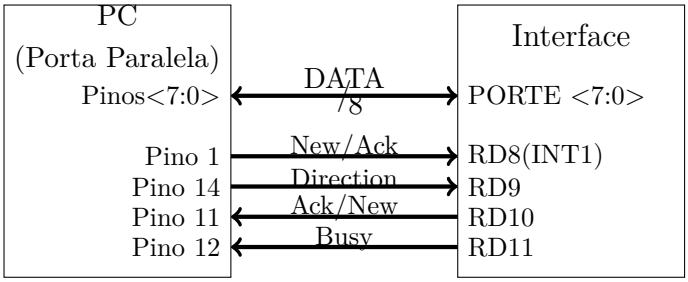


Figura 5.10: Esquema da ligação por porta paralela

Os diagramas temporais das transações de leitura e escrita do barramento de dados por

parte da interface podem ser vistas na figura 5.11. De notar que os sinais *New/Ack* e *Ack/New* deverão ser interpretados como a parte do nome a negrito.

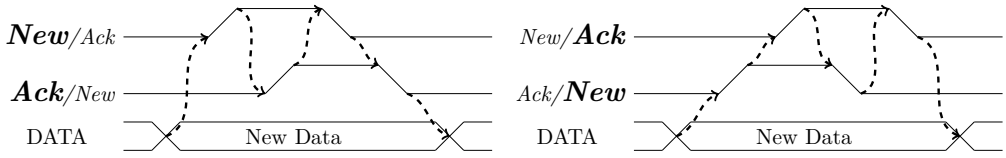


Figura 5.11: Negociação da leitura (esquerda) e escrita (direita)

Analisando a figura 5.11 é possível ver que o protocolo implementa transferências completamente *interlocked*. Antes de a interface efetuar uma leitura do barramento de dados o PC coloca os novos dados no barramento e leva o sinal *New/Ack* ao nível lógico '1', ativando uma interrupção na interface. Durante a interrupção a interface copia o *byte* presente no barramento de dados para o *buffer* de receção e informa de seguida o PC de que terminou a leitura ao levar o sinal *Ack/New* ao nível lógico '1'.

Os passos seguintes servem apenas para colocar os sinais ao seu nível lógico original, após a conclusão destes passos os dados no barramento não deverão ser considerados como válidos.

Novamente na figura 5.11 podemos ver que quando a interface escreve no barramento de dados as negociações se processam de forma muito similar à leitura. A única diferença é o facto de os sinais trocarem entre si as suas funções. Isto acontece porque a ligação dos sinais de controlo é exclusivamente unidirecional ao contrário do barramento de dados.

Estas negociações não são as únicas necessárias para a comunicação entre o PC e a interface visto que a escrita e a leitura não podem ser intercaladas sem que primeiro se inverta o sentido do barramento de dados. É, por isso, também necessário um processo de negociação para esse efeito.

A figura 5.12 apresenta o processo de negociação para a inversão do barramento de dados. As letras *A*, *B*, *C* e *D* representam fases durante as quais alguns acontecimentos a explicar de seguida se processam.

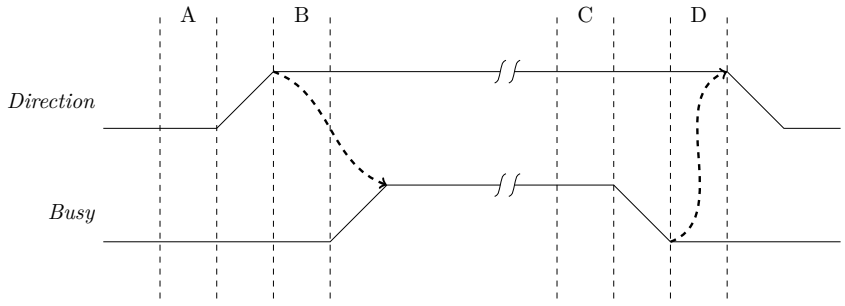


Figura 5.12: Negociação da direção do barramento de dados

Durante a fase *A* o PC, que é o *Master* da ligação, começa por enviar um comando com o pedido de amostragem de entradas analógicas. Logo de seguida configura o seu porto de dados como entrada e por fim leva o sinal *Direction* ao nível lógico '1' de forma a informar a interface de que está pronto para a inversão do sentido do barramento de dados.

Em resposta ao que sucede durante a fase *A* e após ser processado o comando requisitando a leitura de entradas analógicas, a interface irá, durante a fase *B*, desabilitar as interrupções

associadas ao pino associado ao sinal *New/Ack*, de seguida configura o porto de dados como saída e por fim leva o sinal *Busy* a nível lógico '1', ficando o sentido do barramento de dados definido da interface para o PC.

Os passos que ocorrem durante as fases *C* e *D* servem para restituir o sentido do barramento de dados de volta ao seu estado padrão, ou seja, do PC para a interface.

No decorrer da fase *C* a interface envia o último *byte* de dados e configura o porto de dados como entrada. De seguida habilita as interrupções associadas ao sinal *New/Ack* e por último leva o nível lógico do sinal *Busy* a '0', de forma a que o PC saiba que chegou ao fim a transmissão de dados por parte da interface.

Durante a fase *D*, que conclui a transação, o PC reconfigura o seu porto de dados como saída e leva o nível lógico do sinal *Direction* a '0'.

Estão, portanto, explicados todos os processos negociais que tornam a ligação por porta paralela possível. Para acesso à porta paralela no lado do PC, foi utilizada a biblioteca *PARAPIN*[35].

5.3 Medidas e Validação

Com o objetivo de avaliar a qualidade e desempenho da interface desenvolvida efetuaram-se análises de desempenho e análises qualitativas. Esta secção começa por apresentar os resultados das análises de desempenho, seguindo-se a apresentação das análises qualitativas e no fim apresentam-se as conclusões sobre os resultados obtidos.

Na figura 5.13 é apresentada a placa da interface A-D e D-A utilizada.

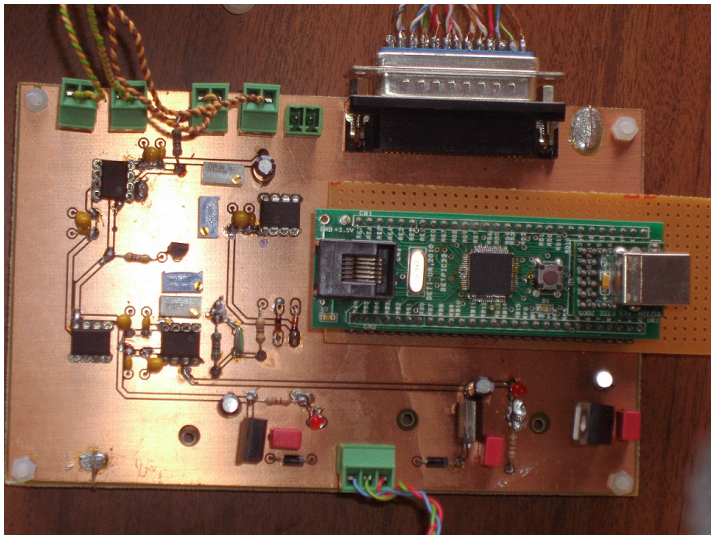


Figura 5.13: Placa da interface A-D e D-A

5.3.1 Análise de desempenho

As análises de desempenho consistiram em medições de latência desde do momento em que um comando é enviado pelo PC até ao momento em que a ação correspondente é concluída.

A latência da maioria dos comandos foi medida no micro-controlador. A única exceção é o comando de leitura, cuja latência foi medida pelo PC.

As medições foram realizadas em 4 cenários diferentes. Os diferentes cenários resultam da combinação dos diferentes tipos de ligação com dois algoritmos de escalonamento de processos do lado do PC. Os algoritmos de escalonamento são o algoritmo de escalonamento genérico do *Linux*, identificado como *Linux* nas tabelas seguintes, e o escalonamento de tempo-real *SCHED_FIFO* da API de POSIX, identificado como *Linux RT*.

O número de vezes que uma medição é repetida num cenário depende do tipo de ligação. Comandos enviados pela ligação de porta paralela foram repetidos 500000 vezes enquanto comandos enviados pela ligação de porta série virtual foram repetidos 50000 vezes. O número elevado de repetições justifica-se pela necessidade de caracterizar a latência o melhor possível e a diferença entre o número de repetições para os dois tipos de ligação justifica-se por serem o suficiente para que em cada tipo de ligação os dados fossem consistentes.

Como as medidas efetuadas do lado do PC e o funcionamento geral do sistema estão sujeitos ao *jitter* e à latência que o funcionamento do sistema operativo impõe aos processos executados, apresentam-se na tabela seguinte as principais características do PC utilizado.

Componente	Modelo
Processador	Intel Pentium(R) 4 CPU 3.00GHz
Gráfica	GeForce4 MX 440 AGP 8x
Memória	1GB
Motherboard	ASUS P4P800S-X (Intel ICH5/ICH5R chipset)
Sistema Operativo	Ubuntu 10.04 com kernel Linux 2.6.32-35

5.3.1.1 *READ*

Quando o utilizador ordena um comando *READ*, começa por enviar uma trama de 5 *bytes* do PC para a interface, requisitando uma amostra do canal indicado. Quando a amostragem é concluída a interface responde ao PC com uma trama de 9 *bytes*, a qual contém os dados requisitados. Na figura 5.14 apresenta-se o diagrama que descreve como as medições foram realizadas, medindo o intervalo de tempo entre o momento em que o comando é enviado e o momento em que a resposta é recebida.

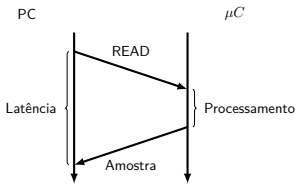


Figura 5.14: Medições de latência das leituras

A figura 5.15 e a tabela 5.2 caracterizam a latência para os diferentes cenários descritos acima.

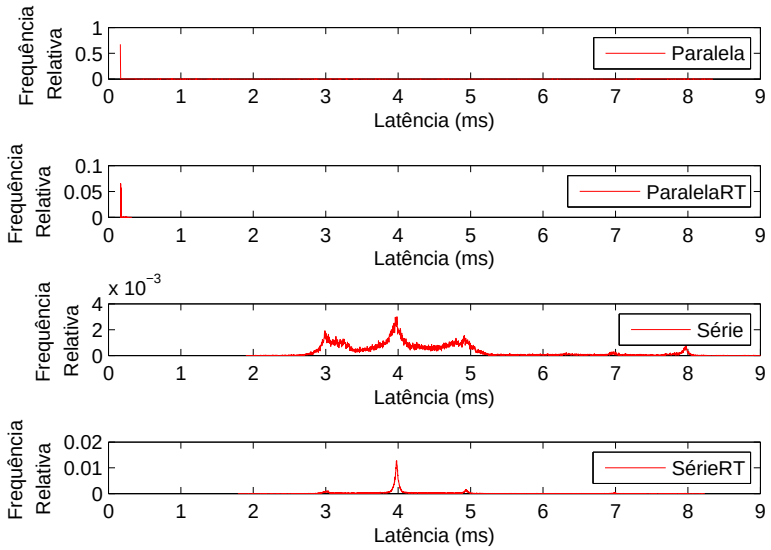


Figura 5.15: Latência do comando *READ* em diferentes cenários.

		Média (ms)	Desvio Padrão (ms)	Mínimo (ms)	Máximo (ms)
Paralela	Linux	258.6	728.7	164.3	8345.9
	Linux RT	174.2	5.8	167.1	322.9
Série	Linux	4240.8	1104.2	1898.0	12807.0
	Linux RT	4081.2	679.4	1789.8	8233.1

Tabela 5.2: Caracterização da latência do comando READ

Para este comando em particular, a ligação por porta paralela é a que apresenta os melhores tempos e é também a que beneficia mais com o escalonamento de tempo-real.

Em média o comando *READ* transferido por porta paralela é executado 16.4 vezes mais rápido que quando transferido por porta série usando o escalonamento genérico e cerca de 23.4 vezes rápido com o escalonamento de tempo-real.

A ligação por porta paralela é também a que apresenta o melhor comportamento no pior caso, tendo um pior caso cerca de 1.5 vezes menor que a ligação por porta série virtual com o escalonamento genérico e 25.5 vezes melhor comportamento com escalonamento de tempo-real.

Na figura 5.16 pode ver-se o gráfico da frequência cumulativa do comando *READ* para os diferentes cenários descritos.

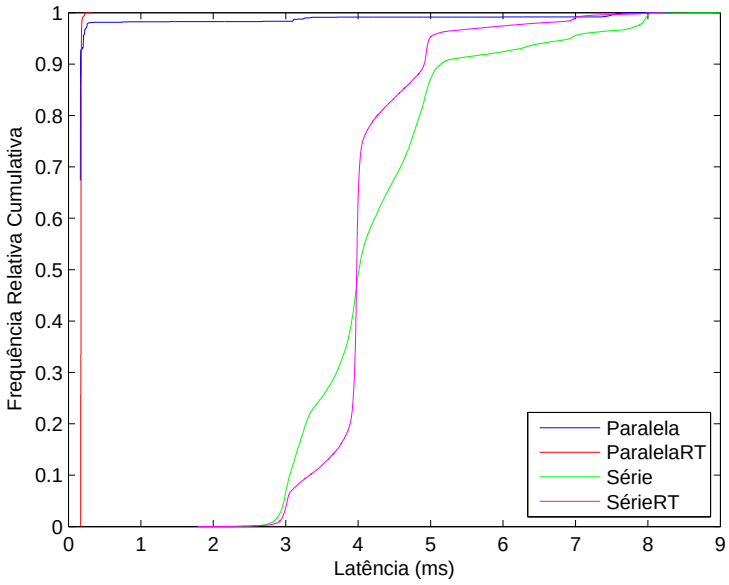


Figura 5.16: Frequência cumulativa do comando *READ* para diferentes cenários.

Para os cenários indicados, os valores de latência abaixo dos quais se conclui 99% das execuções do comando *READ* são apresentados na tabela 5.3.

	Paralela (ms)	ParalelaRT (ms)	Série (ms)	SérieRT (ms)
Latência	3.33	0.20	7.97	7.00

Tabela 5.3: Valores de latência para os quais a frequência relativa cumulativa do comando *READ* é 99%

5.3.1.2 *WRITE*

O comando *WRITE* enviado nesta experiência é composto por uma única trama de 9 *bytes* indicando a saída analógica e o valor a escrever nela. Na figura 5.17 apresenta-se um diagrama que representa o modo como as medições de latência para os comandos *WRITE*, *FILTER*, *RESET* e *TEST* foram realizadas, a latência corresponde ao intervalo de tempo entre o momento em que o comando é enviado pelo PC e o momento em que é executado pela interface. Este procedimento de medida foi o mesmo utilizado nos comandos seguintes.

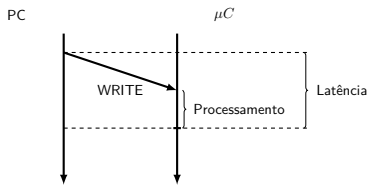


Figura 5.17: Medições de latência dos comandos *WRITE*, *FILTER*, *RESET* e *TEST*

A caracterização da latência para este comando está apresentada na figura 5.18 e na tabela

5.4.

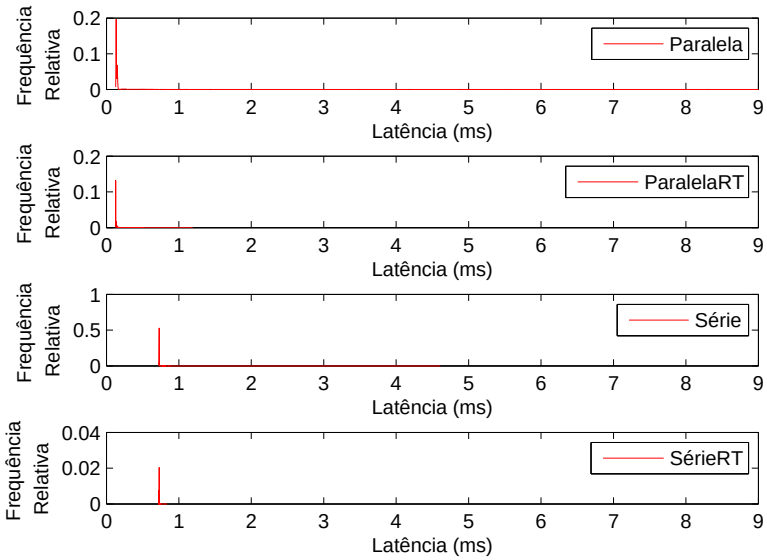


Figura 5.18: Latência do comando *WRITE* em diferentes cenários.

		Média (ms)	Desvio Padrão (ms)	Mínimo (ms)	Máximo (ms)
Paralela	Linux	145.6	42.6	127.7	10038.6
	Linux RT	136.7	16.4	126.9	1192.0
Série	Linux	728.3	18.1	723.6	4608.7
	Linux RT	728.3	1.4	723.8	839.0

Tabela 5.4: Caracterização da latência do comando WRITE

Mais uma vez é a ligação por porta paralela que apresenta os melhores tempos médios, mas desta vez é a que apresenta melhorias menores com o uso de escalonamento de tempo-real.

O tempo médio para a ligação por porta paralela é cerca de 5 vezes menor com o escalonamento genérico e cerca de 5.3 vezes menor com o escalonamento de tempo-real.

Já o pior caso é cerca de 2.2 vezes maior na porta paralela com escalonamento genérico e cerca de 1.4 vezes superior com escalonamento de tempo-real.

Na figura 5.19 pode ver-se o gráfico da frequência relativa cumulativa do comando *WRITE* e na tabela 5.5 os valores de latência para os quais a frequência relativa cumulativa é igual a 99%.

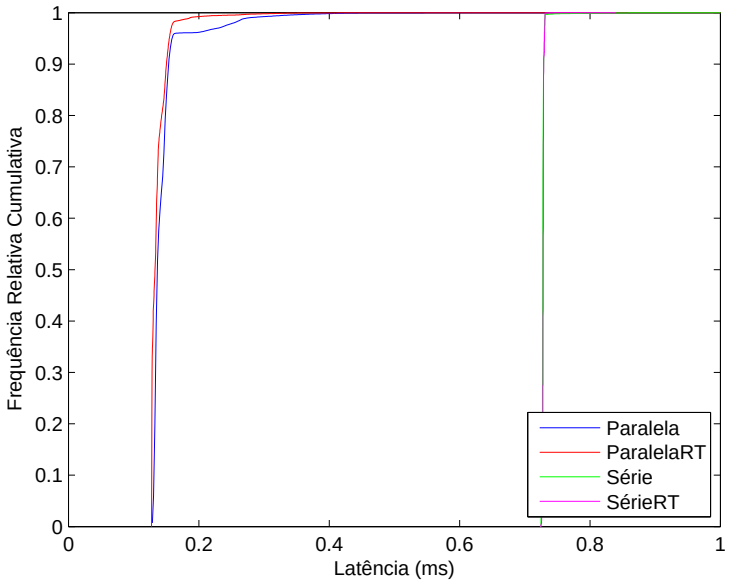


Figura 5.19: Frequência cumulativa do comando *WRITE* para diferentes cenários.

	Paralela (ms)	ParalelaRT (ms)	Série (ms)	SérieRT (ms)
Latência	0.28	0.19	0.73	0.73

Tabela 5.5: Valores de latência para os quais a frequência relativa cumulativa do comando *WRITE* é 99%

5.3.1.3 *FILTER*

Tal como o comando o *WRITE* este comando tem uma trama de 9 *bytes* no qual seguem os parâmetros para configurar os filtros digitais da interface. A figura 5.20 e a tabela 5.6 apresentam os dados que caracterizam a latência deste comando.

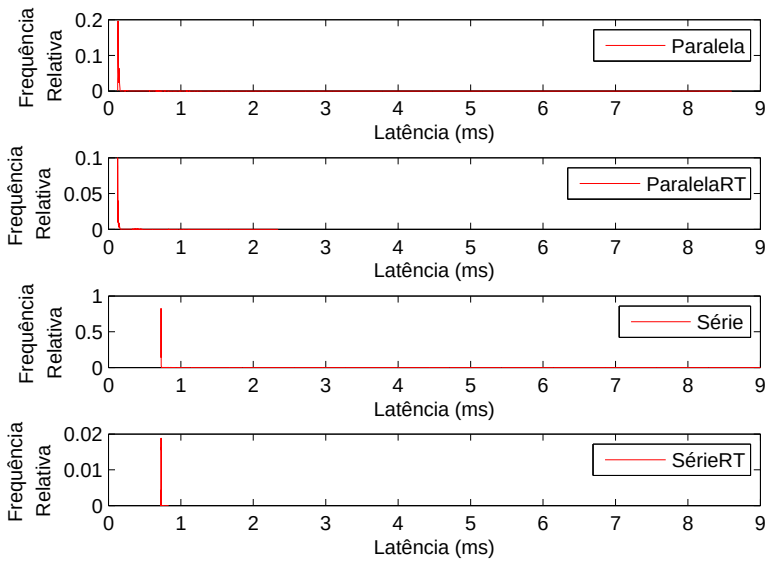


Figura 5.20: Latência do comando *FILTER* em diferentes cenários.

		Média (ms)	Desvio Padrão (ms)	Mínimo (ms)	Máximo (ms)
Paralela	Linux	144.3	40.8	125.8	8603.2
	Linux RT	135.0	17.0	125.6	2342.2
Série	Linux	727.3	84.8	722.9	12685.9
	Linux RT	726.3	1.3	722.7	829.9

Tabela 5.6: Caracterização da latência do comando FILTER

Com um comportamento muito similar ao do comando *WRITE*, os resultados para o comando *FILTER* também demonstram que a ligação por porta paralela apresenta melhores tempos médios e que a ligação por porta série beneficiou mais com o escalonamento de tempo-real.

Desta vez o pior caso para escalonamento genérico é cerca de 1.5 vezes superior para a ligação série e para o escalonamento de tempo-real o pior caso é cerca de 2.8 vezes maior na porta paralela.

Na figura 5.21 pode ver-se o gráfico da frequência relativa cumulativa do comando *FILTER* e na tabela 5.7 os valores de latência para os quais a frequência relativa cumulativa é igual a 99%.

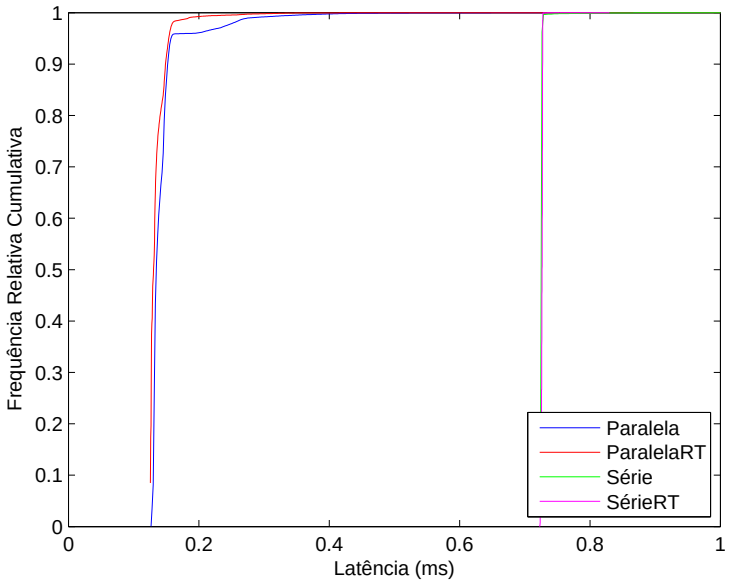


Figura 5.21: Frequência cumulativa do comando *FILTER* para diferentes cenários.

	Paralela (ms)	ParalelaRT (ms)	Série (ms)	SérieRT (ms)
Latência	0.28	0.18	0.73	0.73

Tabela 5.7: Valores de latência para os quais a frequência relativa cumulativa do comando *FILTER* é 99%

5.3.1.4 *RESET*

O comando *RESET* é enviado através de uma trama composta por 5 *bytes*, uma trama sem campo de dados. A latência do comando é caracterizada na figura 5.22 e na tabela 5.8.

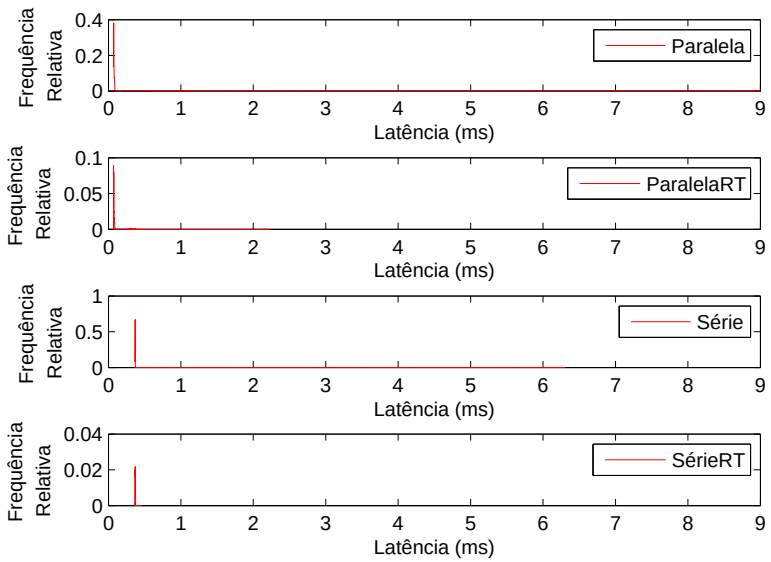


Figura 5.22: Latência do comando *RESET* em diferentes cenários.

		Média (ms)	Desvio Padrão (ms)	Mínimo (ms)	Máximo (ms)
Paralela	Linux	79.5	61.6	69.4	13177.0
	Linux RT	76.9	12.7	69.4	2226.9
Série	Linux	368.7	27.8	365.6	6304.6
	Linux RT	368.2	1.1	365.3	462.7

Tabela 5.8: Caracterização da latência do comando RESET

Como esperado este comando apresenta tempos inferiores aos dos comando anteriores. Qualquer das ligações beneficia da utilização de escalonamento de tempo-real, mas a principal beneficiada é a ligação por porta série.

A ligação por porta paralela apresenta valores mais elevados para o pior caso em ambos os algoritmos de escalonamento. Sendo que o valor para o escalonamento genérico o pior caso é 2.1 vezes superior e para o escalonamento de tempo-real é cerca de 4.8 vezes maior.

Na figura 5.23 pode ver-se o gráfico da frequência relativa cumulativa do comando *RESET* e na tabela 5.9 os valores de latência para os quais a frequência relativa cumulativa é igual a 99%.

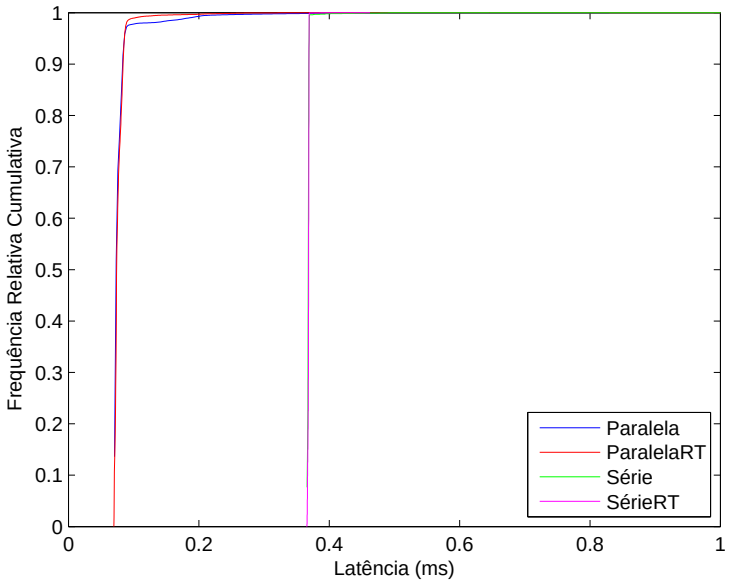


Figura 5.23: Frequência cumulativa do comando *RESET* para diferentes cenários.

	Paralela (ms)	ParalelaRT (ms)	Série (ms)	SérieRT (ms)
Latência	0.19	0.10	0.37	0.37

Tabela 5.9: Valores de latência para os quais a frequência relativa cumulativa do comando *RESET* é 99%

5.3.1.5 *TEST*

Este comando, tal como o comando *RESET*, é um comando cujo a trama não possui campo de dados e é composta por um total de 5 *bytes*. A sua caracterização apresenta-se na figura 5.24 e tabela 5.10.

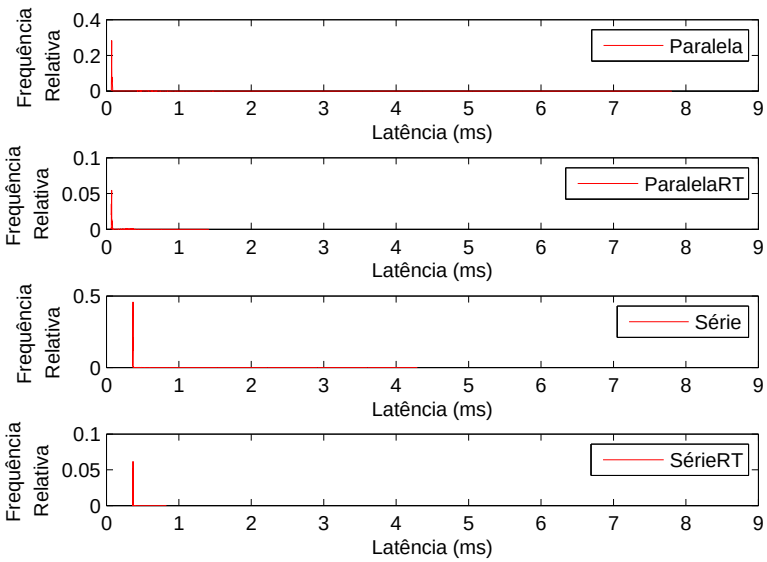


Figura 5.24: Latência do comando *TEST* em diferentes cenários.

		Média (ms)	Desvio Padrão (ms)	Mínimo (ms)	Máximo (ms)
Paralela	Linux	77.5	26.6	66.1	7805.6
	Linux RT	74.5	12.6	65.5	1414.6
Série	Linux	366.0	19.6	362.9	4292.0
	Linux RT	365.6	2.4	362.8	826.4

Tabela 5.10: Caracterização da latência do comando TEST

Os tempos de latência médios são cerca de 4.75 vezes menores para a porta paralela em relação à porta série. Ambas as ligações apresentam melhorias com o uso de escalonamento em tempo-real, sendo essa melhoria muito mais significativa para a ligação de porta série.

O pior caso apresenta mais uma vez os maiores valores com a porta paralela. Sendo que com o escalonamento genérico o pior caso é cerca de 1.8 vezes superior ao da ligação por porta série e com o escalonamento de tempo-real o pior caso é cerca de 1.7 vezes superior.

Na figura 5.25 pode ver-se o gráfico da frequência relativa cumulativa do comando *WRITE* e na tabela 5.11 os valores de latência para os quais a frequência relativa cumulativa é igual a 99%.

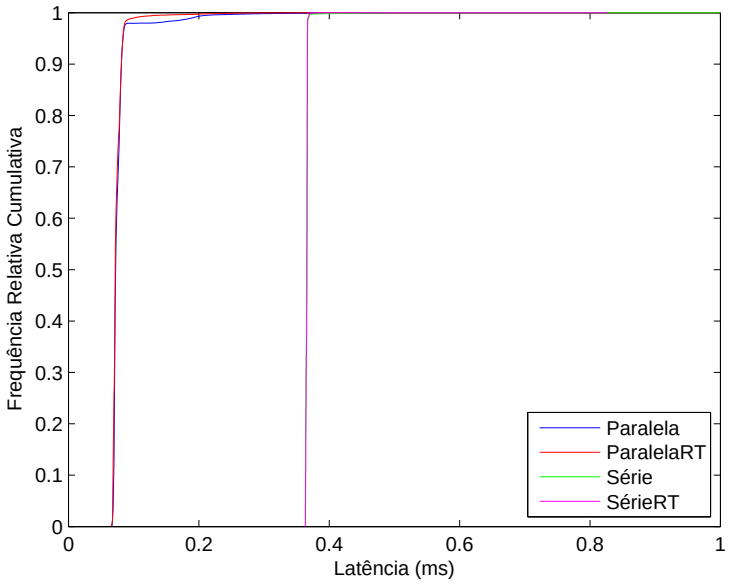


Figura 5.25: Frequência cumulativa do comando *TEST* para diferentes cenários.

	Paralela (ms)	ParalelaRT (ms)	Série (ms)	SérieRT (ms)
Latência	0.19	0.10	0.37	0.37

Tabela 5.11: Valores de latência para os quais a frequência relativa cumulativa do comando *TEST* é 99%

5.3.2 Análise funcional

A análise funcional consistiu em comparar os valores escritos pela interface com os valores medidos nas saídas analógicas e comparar os valores medidos nas entradas analógicas com os valores lidos pela interface.

Para esta análise acoplou-se um circuito externo à interface, ficando este entre a entrada e a saída da mesma. De cada vez que um valor é escrito no conversor *D-A* medem-se os valores de voltagem à saída analógica e à entrada analógica e regista-se o valor lido pelo conversor *A-D*.

Os gráficos que representam essas comparações podem ser consultados na figura 5.26 e os dados registados durante as medições estão presentes na tabela 5.12.

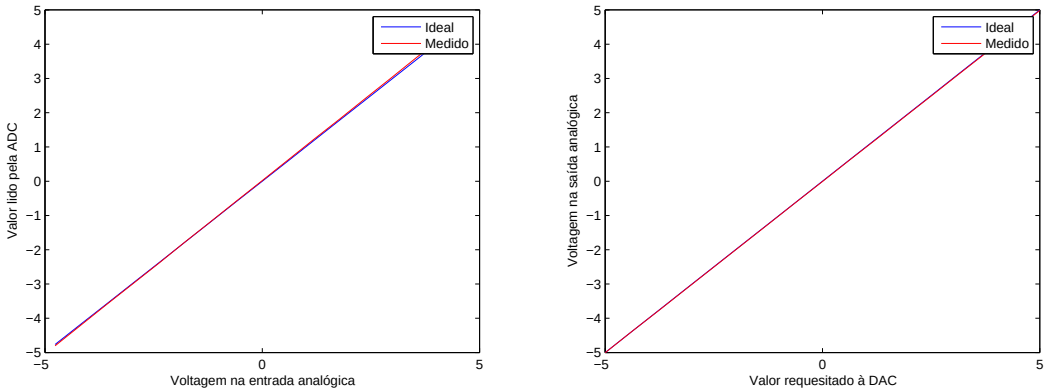


Figura 5.26: Leitura de valores (esquerda) e escrita de valores (direita)

Valor *D-A* (V)	Saída Analógica (V)	Entrada Analógica (V)	Valor *A-D* (V)
-5.0000	-5.0033	-4.7658	-4.8045
-4.3744	-4.3769	-4.1690	-4.1984
-3.7488	-3.7550	-3.5766	-3.5924
-3.1232	-3.1280	-2.9797	-2.9961
-2.4976	-2.5042	-2.3855	-2.3998
-1.8719	-1.8758	-1.7870	-1.7840
-1.2463	-1.2540	-1.1949	-1.1877
-0.6207	-0.6256	-0.5965	-0.5816
0.0049	-0.0041	-0.0048	0.0147
0.6305	0.6242	0.5934	0.6207
1.2561	1.2429	1.1827	1.2170
1.8817	1.8704	1.7800	1.8133
2.5073	2.4950	2.3747	2.4194
3.1329	3.1182	2.9680	3.0156
3.7586	3.7419	3.5620	3.6217
4.3842	4.3723	4.1624	4.2278
5.0000	4.9865	4.7472	4.8143

Tabela 5.12: Medições da análise funcional

A tabela 5.13 apresenta os valores de erros DNL e INL para a interface A-D e D-A calculados a partir dos dados recolhidos.

	DNL (%)	INL (%)
A-D	0.21	0.67
D-A	0.11	0.17

Tabela 5.13: Valores de DNL e INL para a interface A-D e D-A

5.3.3 Análise global

A análise dos dados leva a concluir que a interface respeita todos os requisitos do projeto.

Confirma-se que a ligação por porta paralela é a ligação de preferência para utilizadores mais exigentes, sendo de particular interesse usando escalonamento de tempo-real. Se considerarmos os dados das tabelas 5.3 e 5.5 e considerarmos também que o controlo de um sistema requer uma leitura e uma escrita por ciclo, então a interface poderá ser usada com uma frequência de amostragem de até 277Hz com escalonamento de tempo genérico e de até 2.56KHz com escalonamento *SCHED_FIFO*. É de esperar que melhores tempos se consigam com um sistema operativo de tempo-real que ofereça um melhor controlo temporal.

A ligação por porta série deverá ser operada a frequências consideravelmente inferiores (tabelas tabelas 5.3 e 5.5). Para o escalonamento genérico as frequências de trabalho num sistema de controlo poderão ir até aos 115Hz e até aos 129Hz com o escalonamento *SCHED_FIFO*.

A análise funcional, por sua vez, demonstra que os sinais estão também de acordo com a gama especificada nos requisitos do projeto, variando entre -5 e +5 Volts.

De notar que qualquer das ligações apresenta piores casos que obrigariam um sistema com requisitos de *hard real-time* a operar a frequências consideravelmente mais baixas que as indicadas acima, por isso e ainda que esses valores pudessem melhorar com um sistema operativo de tempo-real a interface deverá ser usada com as devidas precauções.

A justificação para o facto da porta paralela apresentar valores de pior caso superiores para alguns comandos continua por encontrar e ficará para trabalho futuro devido às restrições temporais impostas a esta dissertação.

Capítulo 6

Simulador de Processos contínuos

Os computadores analógicos são utilizados há décadas para a resolução de equações diferenciais, tais como as existentes na simulação de sistemas físicos. A sua utilização permitiu grandes avanços em infraestruturas civis, tecnologia militar e exploração espacial.

No ano de 1931, Vannevar Bush (figura 6.1) finalizou o desenvolvimento do *Differential Analyzer* [1]. O equipamento era um computador mecânico capaz de resolver equações diferenciais até à sexta ordem [39].

Figura 6.1: Vannevar Bush com o *Differential Analyzer*. (Propriedade do Museu do MIT)

A posterior invenção dos amplificadores operacionais (*op-amps*) levou ao desenvolvimento de computadores analógicos eletrónicos, que, no início da década de 1950, representavam a maioria dos computadores analógicos em atividade. No entanto, apesar de mais rápidos e simples, os computadores analógicos eletrónicos eram menos precisos que os computadores analógicos mecânicos. A menor precisão devia-se, em grande parte, à precisão dos componentes e à tecnologia dos *op-amps* da época [40, 41].

Na mesma altura em que começavam a surgir os primeiros computadores analógicos eletrónicos também surgiam os primeiros computadores digitais eletrónicos. Como resolvem equações numericamente, os computadores digitais surgiram como a solução para quem procurava resultados precisos nas suas simulações. Em 1946 era apresentado o ENIAC (*Electronic Numerical Integrator and Computer*) [42].

[1] O desenvolvimento deste equipamento iniciou-se em 1927, tendo sido publicados dois artigos em que o nome dado era *continuous Intergraph* [36, 37]. O trabalho realizado levou a que, no ano seguinte, Vannevar Bush fosse premiado com a *Louis E. Levy Medal* do *Franklin Institute* em Philadelphia [38].

Numa tentativa de juntar o "melhor de dois mundos" começaram a surgir no final da década de 1950 os computadores híbridos, que usavam tanto eletrónica analógica como digital [2].

Com o desenvolvimento tecnológico os computadores digitais acabaram por se tornar rápidos o suficiente para serem a ferramenta mais utilizada na simulação de sistemas físicos, com *software* como *Simulink* ou *Xcos*. No entanto, atualmente os computadores analógicos ainda têm alguma utilidade como simuladores de processos físicos no ensino, projeto e desenvolvimento de controladores de tempo-real [43].

Num sistema de controlo de tempo real, é a dinâmica do processo físico a controlar que impõe as restrições temporais que o sistema computacional precisa respeitar.

O uso de computadores analógicos como simuladores de sistemas justifica-se por serem as únicas ferramentas capazes de simular formalmente um sistema físico, possuindo variáveis contínuas e resolução infinita [44]. Já os simuladores digitais operam num domínio discreto e a um ritmo imposto pelo escalonamento do sistema operativo, não permitindo ao utilizador lidar com aspetos importantes dos sistemas de controlo como a latência de amostragem, atuação e computação.

Os computadores analógicos eletrónicos são uma forma económica e rápida de simular diversos sistemas físicos e avaliar a performance dos controladores desenvolvidos [45].

O simulador de processos contínuos apresentado neste capítulo é uma simplificação dos computadores analógicos eletrónicos e é orientado à simulação de sistemas de 1ª e 2ª ordem. Serve o propósito de testar os algoritmos de controlo desenvolvidos no âmbito desta dissertação. Pretende-se também que o simulador permita ao utilizador variar os parâmetros do sistema simulado com facilidade, pois isso é de particular interesse para o teste de um esquema de controlo adaptativo.

Assim, este capítulo é dedicado ao projeto e desenvolvimento do simulador. Aborda as deduções matemáticas, a implementação e os seus inconvenientes, e por fim como o simulador se comporta quando se variam os seus parâmetros.

6.1 Princípio de funcionamento

6.1.1 Sistema de 1ª ordem

O comportamento dinâmico de um sistema linear de 1º ordem é dado pela equação diferencial apresentada na equação 6.1.

$$\frac{d}{dt}y(t) + \frac{1}{\tau}y(t) = \frac{G_{ss}}{\tau}u(t) \tag{6.1}$$

- Em que:

  τ- constante de tempo do sistema

  G_{ss}- ganho em regime estacionário

No entanto, uma forma mais vantajosa de representar um sistema linear, especialmente quando se tem em vista a sua implementação, é a representação por espaço de estados.

[2] Com eles surgiam também os primeiros conversores analógico-digitais e digital-analógicos.

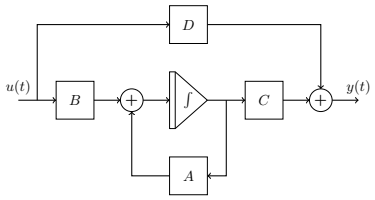


Figura 6.2: Diagrama de representação por espaço de estados

A representação por espaço de estados permite não só relacionar as saídas de um sistema com as suas entradas, como também permite acompanhar o comportamento interno do sistema. Nesta representação a dinâmica do sistema é descrita por duas equações: a equação de estado e a equação de saída. As equações de estado e de saída são representadas, respetivamente, pelas equações 6.2 e 6.3, nas quais $x(t)$ representa o vetor das variáveis de estado do sistema (vetor de estado).

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{6.2}$$

$$y(t) = Cx(t) + Du(t) \tag{6.3}$$

A equação de estado demonstra que o próximo estado do sistema depende do estado atual e das entradas do sistema. A matriz A, designada por matriz de coeficientes, caracteriza a forma como o estado atual influenciará os valores do próximo estado. Já a matriz B, designada por matriz de entrada, caracteriza a forma como as entradas do sistema influenciarão o comportamento interno.

Por sua vez, a equação de saída demonstra a relação entre o vetor de saída e os vetores de estado e de entradas. A matriz C, designada por matriz de saída, relaciona as saídas do sistema com o vetor de estado. A matriz D, designada por matriz de transmissão direta, relaciona as saídas do sistema com as suas entradas.

Assim, o comportamento dinâmico de um sistema de 1ª ordem, que requer apenas uma variável de estado e a qual é o sinal de saída, é descrito pela equação de estado apresentada pela equação 6.4 e pela equação de saída apresentada pela equação 6.5.

$$\dot{x}(t) = -\frac{1}{\tau}x(t) + \frac{G_{ss}}{\tau}u(t) \tag{6.4}$$

$$y(t) = x(t) \tag{6.5}$$

A partir destas equações podemos desenhar um diagrama de blocos que não só representa o sistema de 1ª ordem como também é muito fácil de implementar fisicamente recorrendo a *op-amps*. Como se pode ver na figura 6.3 os diferentes sinais são multiplicados pelo ganho correspondente e somados de forma a obter-se a derivada da variável de estado. Com a integração da derivada obtém-se, então, o sinal de saída do sistema.

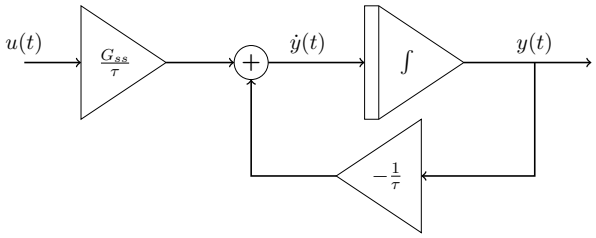


Figura 6.3: Esquema de um sistema de 1ª ordem

6.1.2 Sistema de 2ª ordem

O comportamento dinâmico de um sistema linear de 2ª ordem é dado pela seguinte equação diferencial:

$$\frac{d^2}{dt^2}y(t) + 2\xi\omega_n\frac{d}{dt}y(t) + \omega_n^2 y(t) = G_{ss}\omega_n^2 u(t) \tag{6.6}$$

- Em que:

ω_n-	frequência natural não amortecida
ξ-	coeficiente de amortecimento
G_{ss}-	ganho em regime estacionário

Tal como para o sistema de 1ª ordem, é vantajoso representar o sistema de 2ª ordem em espaço de estados com vista a uma implementação simplificada.

A representação em espaço de estados de um sistema de 2ª ordem possui, pelo menos, duas variáveis de estado. Assim, um vetor de estado possível é apresentado na equação 6.7, este vetor é composto pelo sinal $y(t)$, a saída, e pela sua derivada $\dot{y}(t)$.

$$\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} y(t) \\ \dot{y}(t) \end{bmatrix} \tag{6.7}$$

Com este vetor de estado obtém-se a seguinte representação em espaço de estados para o sistema de 2ª ordem (equação 6.6). A equação de estado e equação de saída são apresentadas, respetivamente, pelas equações 6.8 e 6.9.

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\xi\omega_n \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ G_{ss}\omega_n^2 \end{bmatrix} u(t) \tag{6.8}$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{6.9}$$

A partir das matrizes A, B, C e D, podemos desenhar o diagrama de blocos que representa o sistema de 2ª ordem. O diagrama de blocos é apresentado na figura 6.4.

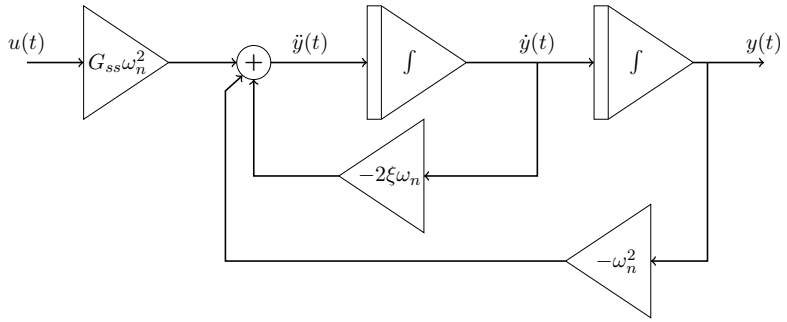


Figura 6.4: Esquema de um sistema de 2ª ordem

Recorrendo a dois *op-amps* em topologia de integradores, facilmente se implementa este sistema. As variáveis de estado $x_1(t)$ e $x_2(t)$ e o sinal de entrada $u(t)$ são multiplicados pelos ganhos correspondentes e somados para se obter $\dot{x}_2(t)$, ou seja $\ddot{y}(t)$. A integração de $\dot{x}_2(t)$ permite-nos obter $x_2(t)$, que, recorde-se, corresponde a $\dot{y}(t)$. E por fim a integração de $x_2(t)$ permite-nos obter $x_1(t)$, que conforme nos indica a equação de saída corresponde a $y(t)$.

6.2 Implementação

A implementação dos simuladores de sistemas de 1ª e 2ª ordem, apresentados respetivamente nas figuras 6.3 e 6.4, poderá ser feita num único circuito eletrónico. Uma sugestão para tal circuito é apresentada na figura 6.5, neste circuito um interruptor (*SWITCH*) permite ao utilizador escolher a topologia do segundo *op-amp OPAMP2* e consequentemente a ordem do sistema.

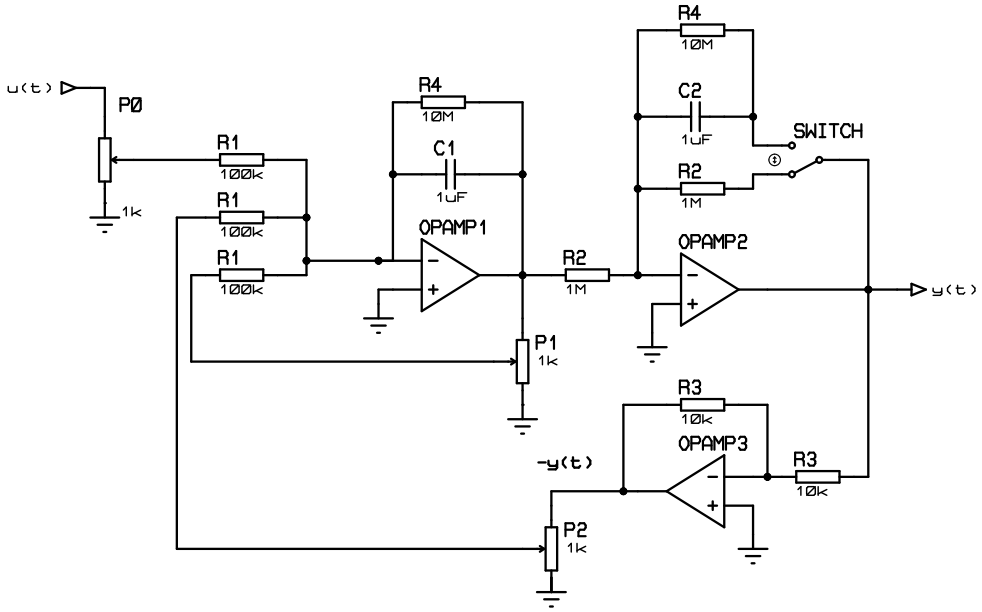


Figura 6.5: Circuito do simulador de sistemas lineares de 1ª e 2ª ordem

Assim, quando o *OPAMP2* está configurado como um inversor o sistema comporta-se como um sistema de 1ª ordem. O primeiro *op-amp* (*OPAMP1*) é um somador integrador e gera o sinal $-y(t)$, o que leva a que à saída de *OPAMP2* surja $y(t)$.

Configurando o *OPAMP2* como integrador obtemos $y(t)$ à saída do circuito e $-\dot{y}(t)$ à saída de *OPAMP1*, ou seja obtemos o circuito de 2ª ordem que se projetou.

As resistências identificadas com R_4 servem o propósito de evitar a saturação dos *opamps.* O efeito que estas resistências têm no comportamento do simulador é demonstrado em 6.2.2.

Na figura 6.6 é apresentada a placa do simulador de processos contínuos.

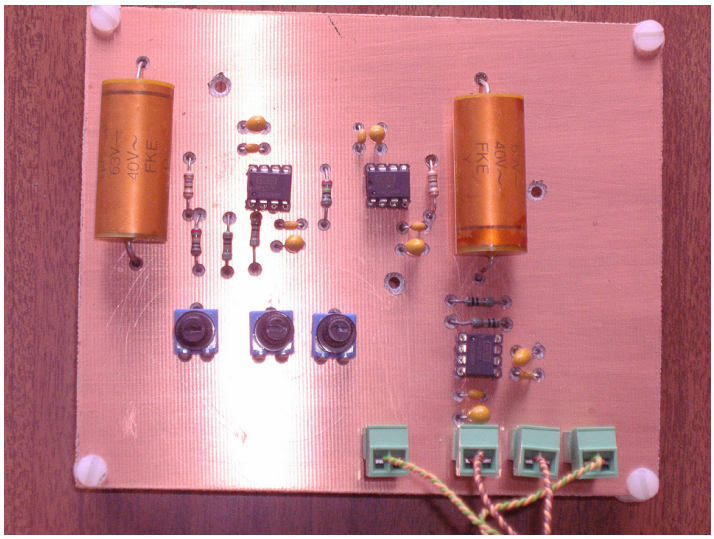


Figura 6.6: Placa do simulador de processos contínuos

6.2.1 Variação de coeficientes

O ajuste dos potenciómetros presentes no circuito da figura 6.5 permite ao utilizador variar os coeficientes do sistema a simular.

A utilização de um potenciómetro como divisor resistivo permite-nos obter no seu ponto médio uma fração do sinal aplicado aos seus terminais, ou seja, permite-nos aplicar ao sinal original um ganho real entre 0 e 1. Para que o ganho aplicado aos sinais seja maior, e consequentemente se aumente a gama de sistemas possíveis de simular, dimensionou-se C_1 e R_1 de forma a que $\frac{1}{C_1 R_1} = 10$. Isto permite que os coeficientes do polinómio característico do sistema de 2ª ordem passem a variar entre 0 e 10, em lugar de entre 0 e 1, e o do sistema de 1ª ordem entre 0 e 20, em lugar de entre 0 e 2.

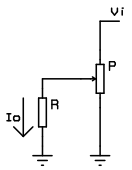


Figura 6.7: Circuito para variação de coeficientes

Analisando a figura 6.7 é fácil perceber que a tensão no ponto médio do potenciómetro também depende da corrente I_o. Este efeito é indesejável e poderia ser evitado ligando o ponto médio do potenciómetro à entrada de um seguidor de tensão. No entanto, tendo o cuidado de dimensionar corretamente os componentes do circuito de variação de coeficientes, a não-linearidade do circuito torna-se negligenciável.

Se considerarmos αP como sendo a resistência entre o ponto médio e a massa do potenciómetro, podemos escrever a seguinte expressão de I_o em função de V_i:

$$I_o = \frac{1}{R}\frac{\alpha P \| R}{\alpha P \| R + (1-\alpha)P} V_i = \frac{\alpha}{R}\frac{R}{R + \alpha(1-\alpha)P} V_i \tag{6.10}$$

Para que a não-linearidade da equação seja negligenciável temos que garantir que $R \gg \alpha(1-\alpha)P$ de forma a que $\frac{R}{R+\alpha(1-\alpha)P}$ esteja o mais perto possível da unidade.

Assim, como $\alpha(1-\alpha)$ tem um máximo de 0.25 para $\alpha = 0.5$ e $R = 100k\Omega$ só resta definir o valor de P em função da não linearidade que se pretende para o circuito. A figura 6.8 mostra a relação entre a não-linearidade e a resistência total do potenciómetro quando $R = 100k\Omega$. Para o circuito da figura 6.5 foram selecionados potenciómetros de $1k\Omega$ majorando a não linearidade a cerca de 0.25%.

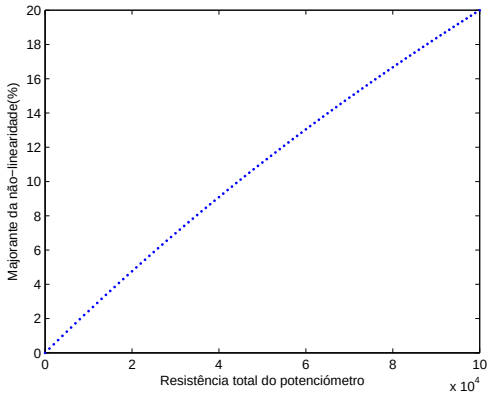


Figura 6.8: Não linearidade em função do valor da resistência total do potenciómetro ($R = 100k\Omega$)

6.2.2 Os polos dos sistemas em função da posição dos pontos médios dos potenciómetros

6.2.2.1 Sistema de 1ª ordem

Para se conhecer de que forma o circuito apresentado na figura 6.5 se comporta como um sistema de 1ª ordem é útil redefinir a equação 6.1 em função do ajuste dos potenciómetros.

Assim, considerando α como sendo a relação entre a resistência entre o ponto médio de um potenciómetro e o potencial zero e entre a sua resistência total podemos redefinir a equação 6.1 da seguinte forma.

$$\frac{d}{dt}y(t) + \frac{\frac{R_1}{R_4} + (\alpha_1 + \alpha_2)}{C_1 R_1} y(t) = \frac{\alpha_0}{C_1 R_1} u(t) \tag{6.11}$$

Comparando as equações 6.1 e 6.11 podemos deduzir os parâmetros do sistema em função de α, tal como se apresenta nas equações 6.12 e 6.13.

$$\tau = \frac{C_1 R_1}{\frac{R_1}{R_4} + (\alpha_1 + \alpha_2)} \tag{6.12}$$

$$G_{ss} = \frac{\alpha_0}{\frac{R_1}{R_4} + (\alpha_1 + \alpha_2)} \tag{6.13}$$

É assim possível para este circuito simular um sistema de 1ª ordem com um ganho estático (G_{ss}) de qualquer valor real entre 0 a 100 e uma constante de tempo (τ) que pode variar entre 0.0498 e 10.

O polo do sistema é dado pela a equação 6.14 e implica que este circuito é capaz de simular um sistema de 1ª ordem com um polo entre 0.1 e 20.1 $^{rad}/_{s}$.

$$Polo = -\frac{\frac{R_1}{R_4} + (\alpha_1 + \alpha_2)}{C_1 R_1} \tag{6.14}$$

6.2.2.2 Sistema de 2ª ordem

Tal como para o sistema de 1ª ordem, comecemos por redefinir a equação 6.6 de forma a que a dinâmica do sistema seja expressa em função da posição do ponto médio dos potenciómetros. A equação 6.15 é o resultado dessa redefinição.

$$\frac{d^2}{dt^2}y(t) + \frac{R_1R_2R_4(C_1+C_2) + C_2R_2R_4^2\alpha_1}{C_1C_2R_1R_2R_4^2}\frac{d}{dt}y(t) + \frac{R_1R_2 + R_2R_4\alpha_1 + R_4^2\alpha_2}{C_1C_2R_1R_2R_4^2}y(t) = \frac{R_4^2\alpha_0}{C_1C_2R_1R_2R_4^2}u(t) \tag{6.15}$$

Mais uma vez, comparando as equações 6.6 e 6.15 podemos deduzir os valores dos parâmetros ω_n, ξ e G_{ss} do sistema de 2ª ordem em função de α. As equações apresentam-se de seguida.

$$\omega_n = \sqrt{\frac{\frac{R_1R_2}{R_4^2} + \frac{R_2}{R_4}\alpha_1 + \alpha_2}{C_1C_2R_1R_2}} \tag{6.16}$$

$$\xi = \frac{1}{2\sqrt{C_1C_2R_1R_2}} \frac{R_1R_2(C_1+C_2) + C_2R_2R_4\alpha_1}{\sqrt{R_1R_2 + R_2R_4\alpha_1 + R_4^2\alpha_2}} \tag{6.17}$$

$$G_{ss} = \frac{\alpha_0}{\frac{R_1R_2}{R_4^2} + \frac{R_2}{R_4}\alpha_1 + \alpha_2} \tag{6.18}$$

O parâmetro ω_n poderá ter qualquer valor real entre 0.1 e 3.3181 $^{rad}/_{s}$, por sua vez ξ poderá variar entre 0.0301 e 51 e G_{ss} poderá ter qualquer valor real positivo entre 0 e 1000.

Por fim ambos os polos do sistema poderão ser calculados recorrendo à equação 6.19.

$$\begin{aligned} Polo_{1,2} &= -\frac{1}{2}\frac{R_1R_2R_4(C_1+C_2) + C_2R_2R_4^2\alpha_1}{R_4^2R_1R_2C_1C_2} \\ &\pm\sqrt{\frac{1}{4}\left(\frac{R_1R_2R_4(C_1+C_2) + C_2R_2R_4^2\alpha_1}{R_4^2R_1R_2C_1C_2}\right)^2 - \frac{R_1R_2 + R_2R_4\alpha_1 + \alpha_2R_4^2}{R_4^2R_1R_2C_1C_2}} \end{aligned} \tag{6.19}$$

A figura 6.9 apresenta as posições no plano complexo de Laplace que os polos deste sistema de 2ª ordem podem ocupar.

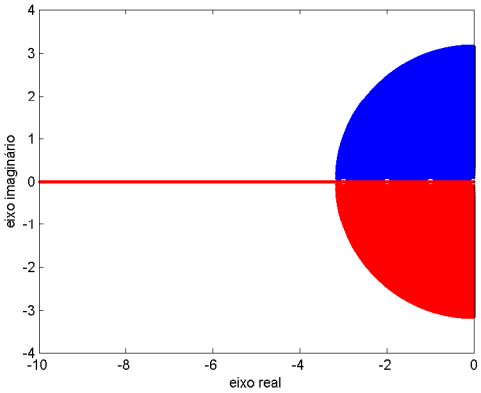


Figura 6.9: Disposição dos polos do sistema de 2ª ordem no domínio de Laplace

Capítulo 7

Controlo por posicionamento de polos com *Xenomai Lab*

Neste capítulo é apresentado o bloco que integra o controlador adaptativo baseado na técnica de posicionamento de polos desenvolvido no âmbito desta dissertação.

Na primeira parte são apresentadas as características do controlador, bem como, a sua interface de configuração.

Na segunda parte são apresentados alguns exemplos de aplicação. Estes exemplos procuram demonstrar a utilidade de algumas características do controlador.

Na terceira parte é apresentada uma análise à latência de execução do controlador desenvolvido.

7.1 Integração no *Xenomai Lab*

O bloco onde se integra o controlador adaptativo desenvolvido tem o nome de *AdatpCtrl*.

Para o algoritmo de identificação de sistemas o utilizador pode optar entre o algoritmo de mínimos quadrados recursivo com fator de esquecimento direcional e o algoritmo de mínimos quadrados recursivo com fator de esquecimento exponencial. Para melhorar a qualidade das estimativas, ao vetor de regressores é realizada uma filtragem passa-banda [1].

Esta filtragem permite diminuir o efeito que o ruído de alta-frequência tem na velocidade de convergência e o efeito que as perturbações de baixa-frequência como o *offset* têm nos valores para os quais os valores de estimação convergem [21].

O filtro utilizado no bloco do *AdaptCtrl* é $H_f(z) = \frac{z-1}{A_m(z)A_o(z)}$.

O algoritmo utilizado para a resolução da equação de Diophantine foi o algoritmo de resolução por via matricial, descrito em 4.2.2.

Na figura 7.1 é apresentada a janela de configuração do bloco.

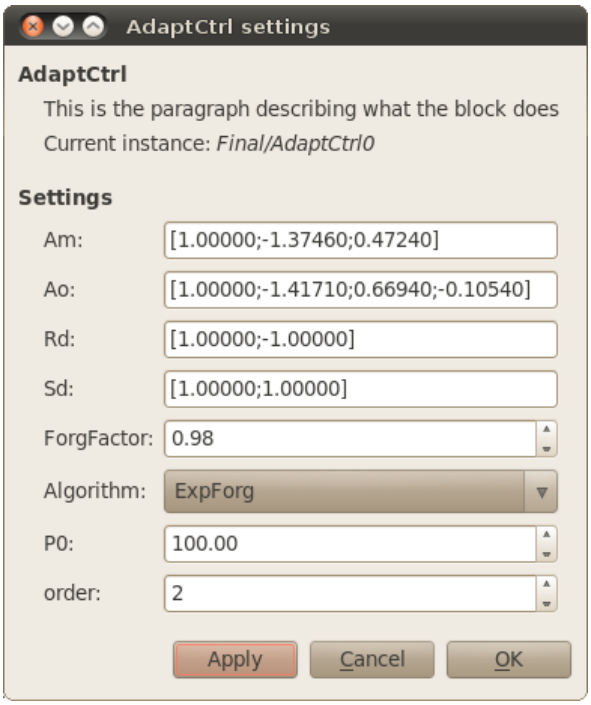


Figura 7.1: Bloco de configuração do bloco do controlador adaptativo

O utilizador deverá indicar a equação característica do sistema em malha fechada (*Ao* e *Am*) e também algumas características do controlador (*Rd* e *Sd*), como a ordem do sistema (*order*), o fator de esquecimento utilizado na estimação de parâmetros (*ForgFactor*), o tipo de fator de esquecimento (*Algorithm*) e valores de inicialização para os coeficientes da diagonal principal da matriz de covariância (*P0*).

O controlador opera no domínio de z. Consequentemente, os polinómios indicados neste bloco deverão ser expressos no mesmo domínio.

Os dados inseridos nos campos *Am*, *Ao*, *Rd* e *Sd* são utilizados no projeto do controlador.

Em *Am* o utilizador deverá indicar o polinómio característico que deseja para o sistema em malha fechada. O polinómio característico deverá ser representado por um vetor coluna composto pelos coeficientes do polinómio.

Em *Ao* deverá ser indicado um vetor coluna com os coeficientes do polinómio observador. A ordem deste polinómio terá que respeitar:

$$deg(A_o) = 2deg(A) + deg(R_d) + deg(S_d) - deg(A_m) - 1$$

Em *Rd* e *Sd* devem indicar-se os fatores desejados para os polinómios R e S, respetivamente.

Os dados inseridos nos campos *ForgFactor*, *Algorithm*, *P0* e *order* são utilizados pelo estimador inerente ao bloco *AdaptCtrl*.

Em *ForgFactor* deve ser indicado o valor do fator de esquecimento.

Em *Algorithm* deve-se escolher o fator de esquecimento a utilizar pelo algoritmo de identificação. Para utilizar fator de esquecimento direcional deve-se escolher *DirForg* e para utilizar fator de esquecimento exponencial deve-se escolher *ExpForg*.

Em *P0* deve ser indicado o valor com o qual deve ser inicializada a diagonal principal da matriz de covariância.

Em *order* deve ser indicada a ordem do sistema a controlar. Este valor é utilizado para dimensionar todos os vetores e matrizes envolvidos na estimação do sistema e cálculo do controlador.

7.2 Exemplos de aplicação

Para demonstrar as capacidades do controlador adaptativo numa situação real e tudo o que esta implica (*offset*, ruído, etc...), bem como a necessidade de algumas das propriedades do controlador, apresentam-se alguns exemplos de aplicação.

O diagrama de blocos utilizado no *Xenomai Lab* é apresentado na figura 7.2. Para todos os exemplos foi utilizado um período de amostragem de $150ms$. Para a comunicação com a interface A-D e D-A foi utilizada a porta paralela.

Nos exemplos, todos os sistemas em malha aberta e em malha fechada são de segunda ordem. A dinâmica de um sistema de segunda ordem é descrita pela equação diferencial:

$$\frac{d^2}{dt^2}y(t) + 2\xi\omega_n\frac{d}{dt}y(t) + \omega_n^2 y(t) = G_{ss}\omega_n^2 u(t)$$

Para todos os exemplos, a dinâmica desejada para o sistema em malha fechada é a de um sistema de segunda ordem com $\omega_n = 2.5$ e $\xi = 1$ e $G_{ss} = 1$. No domínio discreto, e para o período de amostragem indicado, a equação característica é dada por:

$$A_m(z) = z^2 - 1.375z + 0.472$$

As raízes do polinómio observador são todas localizadas em $-2\omega_n = -5^{rad}/_s$. A única diferença entre os polinómios observadores dos vários exemplos é então a sua ordem.

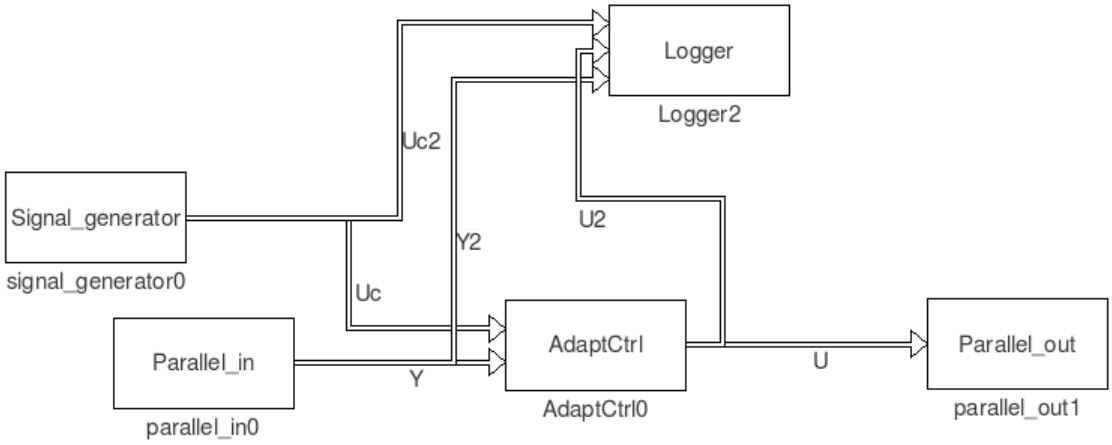


Figura 7.2: Diagrama de Blocos

O bloco *Logger* foi desenvolvido para poder registar os valores dos sinais do sistema. No entanto, não é possível registar as variáveis internas do controlador. Assim, para os gráficos seguintes onde são apresentadas as estimativas dos coeficientes do sistema ao longo do tempo,

os dados apresentados são calculados em *MATLAB* a partir dos valores dos sinais registados, usando os mesmos algoritmos.

7.2.1 Controlador Adaptativo com ação integral

Para o controlador adaptativo deste exemplo definiu-se $R_d = z - 1$, $S_d = 1$, $A_o = z^2 - 0.945z + 0.223$, $ForgFactor = 0.98$ e foi selecionado o fator de esquecimento direcional. O fator requerido para o polinómio R implica que o controlador possuirá ação integral, permitindo anular o erro em regime estacionário.

O simulador de sistemas contínuos foi configurado como um sistema de segunda ordem com $\omega_n = 2.269$, $\xi = 0.301$ e $G_{ss} = 0.960$.

Para o período de amostragem indicado, a mesma dinâmica pode ser descrita pela equação de diferenças seguinte:

$$y(k) - 1.711y(k-1) + 0.815y(k-2) = 0.052u(k-1) + 0.048u(k-2) \tag{7.1}$$

Na figura 7.3 pode ver-se a evolução das estimativas dos coeficientes do sistema ao longo da experiência.

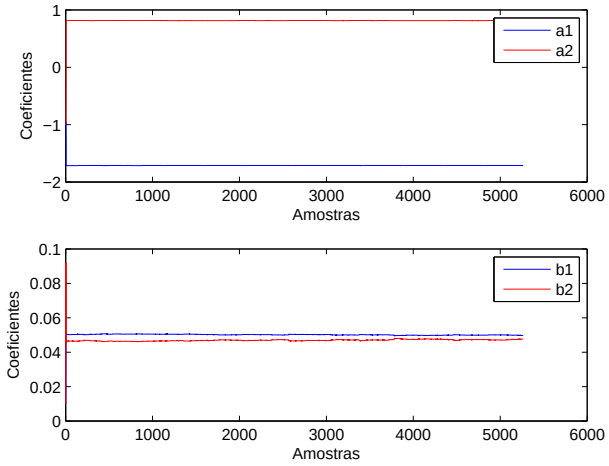


Figura 7.3: Evolução de estimação de coeficientes

Os valores estimados no fim da experiência correspondem aos da equação de diferenças seguinte:

$$y(k) - 1.713y(k-1) + 0.814y(k-2) = 0.050u(k-1) + 0.048u(k-2) \tag{7.2}$$

Na figura 7.4 podem ver-se os sinais de entrada e saída do sistema, sendo U_c o sinal de referência, U o sinal de controlo, Y o sinal de saída do sistema e Y_m o sinal pretendido à saída do sistema. Notar que U é bastante oscilatório quando o sistema está em regime estacionário.

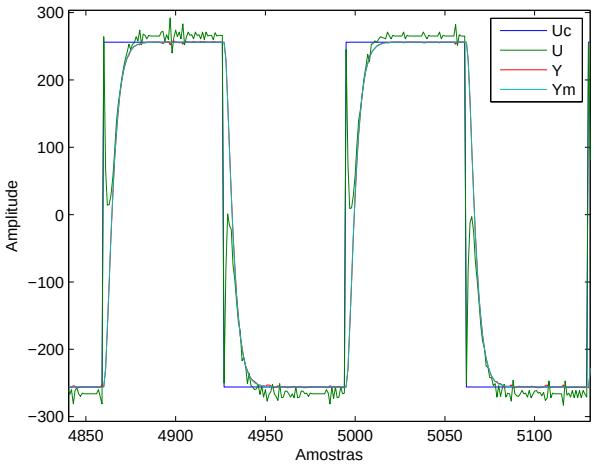


Figura 7.4: Detalhe de sinais

Na figura 7.5 apresenta-se o sinal de erro do sinal de saída do sistema, ou seja, a diferença entre o sinal Y e o sinal Y_m.

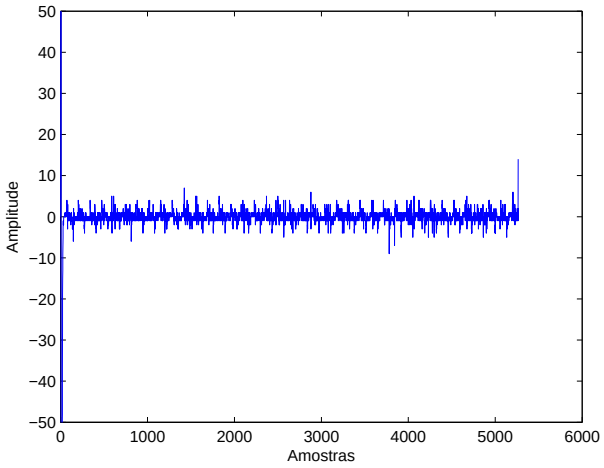


Figura 7.5: Erro à saída do sistema

7.2.2 Controlador Adaptativo com ação integral e fator de atenuação de altas frequências

Para este exemplo definiu-se $R_d = z - 1$, $S_d = z + 1$, $A_o(z) = z^3 - 1.417z^2 + 0.669z - 0.105$, $ForgFactor = 0.98$ e foi selecionado o fator de esquecimento direcional. O fator requerido para o polinómio S permite atenuar as altas-frequências na malha de realimentação e consequentemente atenua também a oscilações de alta-frequência no sinal de controlo [1].

O simulador de sistemas contínuos foi mais uma vez configurado tal como no exemplo 7.2.1.

Na figura 7.6, pode ver-se a evolução das estimativas dos coeficientes.

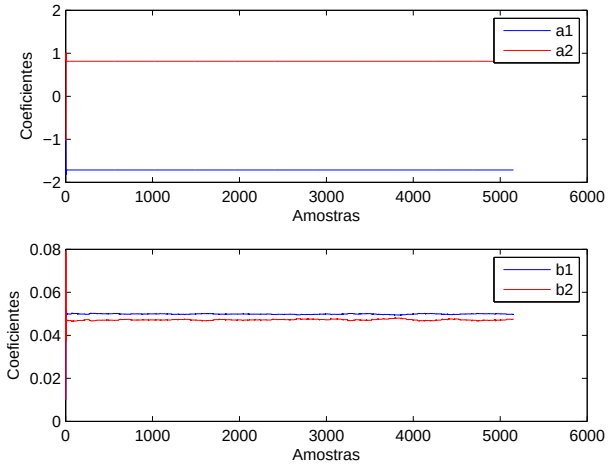


Figura 7.6: Evolução de estimação de coeficientes

No final da experiência os coeficientes estimados pertencem aos da equação de diferenças seguinte:

$$y(k) - 1.713y(k-1) + 0.814y(k-2) = 0.050u(k-1) - 0.048u(k-2) \tag{7.3}$$

Notar na figura 7.7 que o sinal de controlo possui menos oscilação que o sinal de controlo apresentado na figura 7.4.

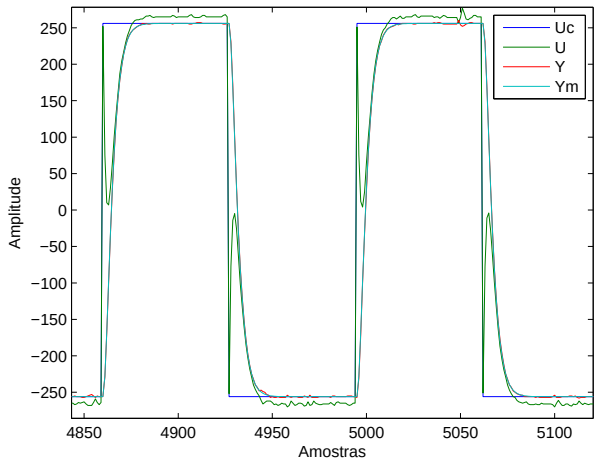


Figura 7.7: Detalhe dos sinais

Na figura 7.8 pode ver-se o sinal de erro a saída dos sistemas.

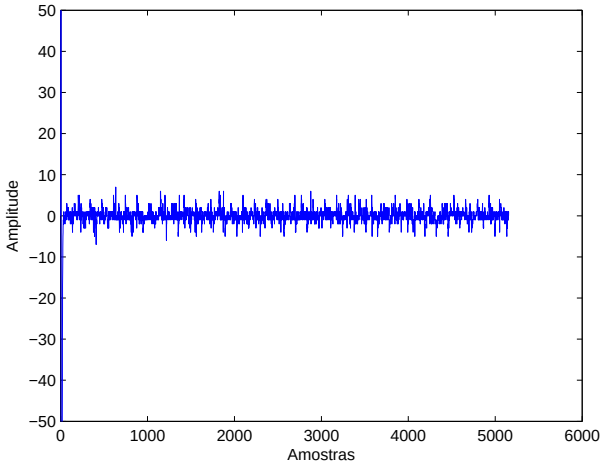


Figura 7.8: Erro do sinal de saída

Este controlador é mais "robusto" que os anteriores, permitindo a compensação de erro estacionário e atenuação de oscilações no sinal de controlo. Por isso, é o controlador mais aconselhável, de entre os apresentados, para utilização genérica.

7.2.3 Controlador Adaptativo e variação de parâmetros (1)

Neste exemplo procura-se demonstrar as capacidades do controlador estimar, em tempo real, os parâmetros de um sistema variante no tempo usando um fator de esquecimento direcional.

Foi usado o mesmo controlador do exemplo 7.2.2.

Para este exemplo, o simulador de sistemas foi inicialmente configurado como um sistema de segunda ordem com $\omega_n = 2.269$, $\xi = 0.301$ e $G_{ss} = 0.960$. Posteriormente, o sistema foi alterado para um sistema de segunda ordem com $\omega_n = 2.941$, $\xi = 0.233$ e $G_{ss} = 0.984$.

Na figura 7.9 pode ver-se a evolução das estimativas, note-se que a alteração do sistema ocorre na amostra 3000.

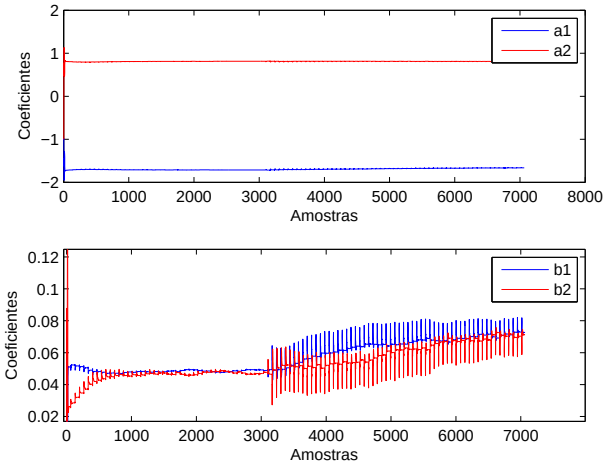


Figura 7.9: Evolução da estimação dos coeficientes

Na figura 7.10 podem ver-se os sinais do sistema antes e depois da mudança dos parâmetros.

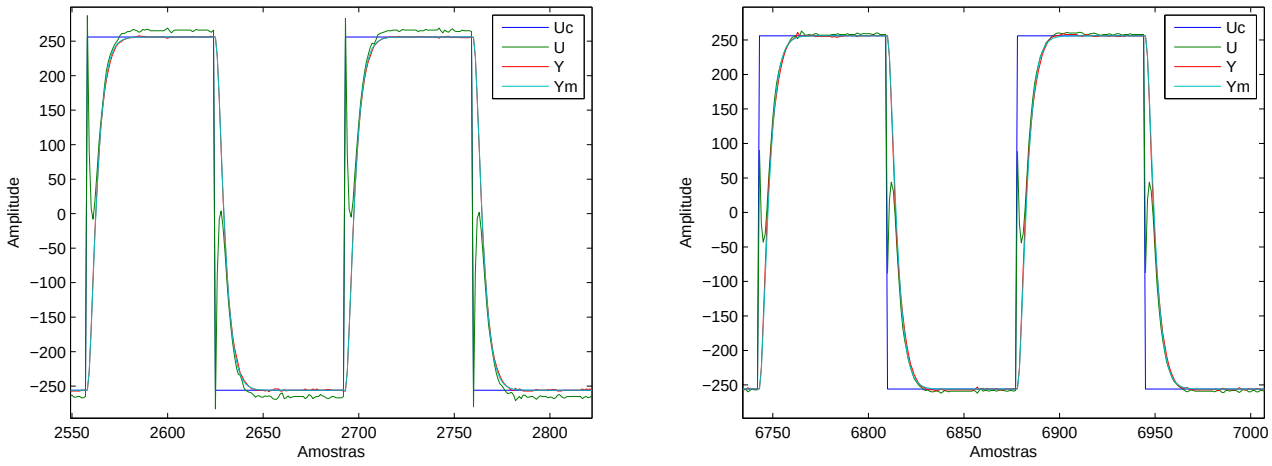


Figura 7.10: Detalhe dos sinais do sistema antes da mudança de parâmetros (esquerda) e depois da mudança de parâmetros (direita)

Na figura 7.11 é apresentado o sinal de erro à saída do sistema.

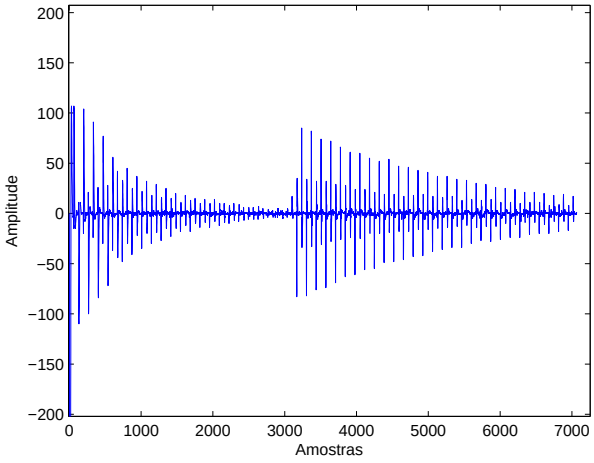


Figura 7.11: Erro à saída do sistema

Como pode ver-se pelos gráficos das figuras 7.10 e 7.11, o controlador foi capaz de se adaptar à mudança dos parâmetros do sistema, tendo o sinal de saída do sistema Y convergido mais uma vez para o sinal desejado Y_m.

Como esperado, e como foi descrito em 4.3.2, o algoritmo de mínimos quadrados recursivo com fator de esquecimento direcional é lento na convergência das estimativas dos coeficientes do sistema quando este sofre variações bruscas.

7.2.4 Controlador Adaptativo e variação de parâmetros (2)

Este exemplo é muito similar ao exemplo 7.2.3, a única diferença relevante está no fator de esquecimento utilizado para a identificação do sistema.

Definiu-se $R_d = z-1$, $S_d = z+1$, $A_o(z) = z^3-1.417z^2+0.669z-0.105$, $ForgFactor = 0.98$ e foi selecionado o fator de esquecimento exponencial.

O simulador de sistemas foi inicialmente configurado como um sistema de segunda ordem com $\omega_n = 2.251$, $\xi = 0.315$ e $G_{ss} = 0.943$. Posteriormente, o sistema foi alterado para um sistema de segunda ordem com $\omega_n = 2.931$, $\xi = 0.242$ e $G_{ss} = 1.043$.

Na figura 7.12 pode ver-se a evolução das estimativas, note-se que a alteração do sistema ocorre na amostra 900.

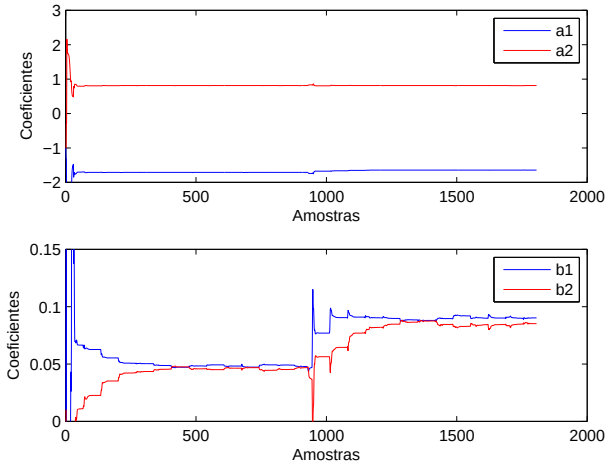


Figura 7.12: Evolução da estimação dos coeficientes

Na figura 7.13 podem ver-se os sinais do sistema antes e depois da mudança dos parâmetros.

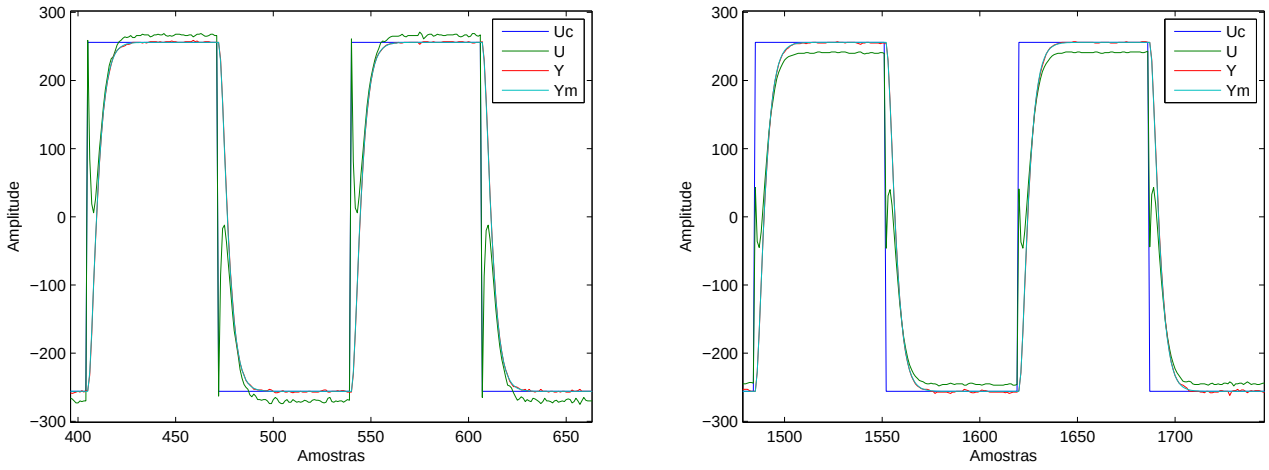


Figura 7.13: Detalhe dos sinais do sistema antes da mudança de parâmetros (esquerda) e depois da mudança de parâmetros (direita)

Na figura 7.14 é apresentado o sinal de erro à saída do sistema.

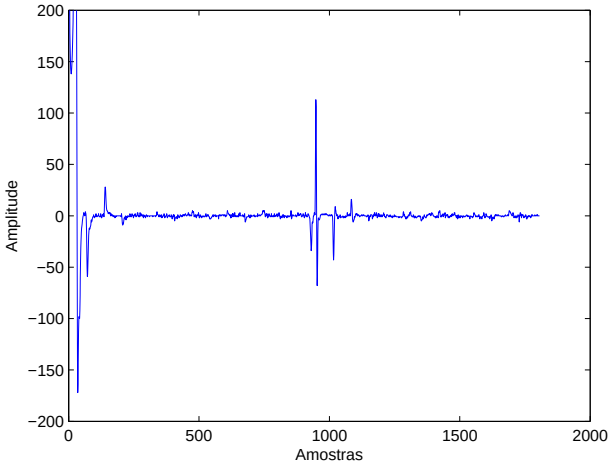


Figura 7.14: Erro à saída do sistema

Tal como para o exemplo 7.2.3, o controlador neste exemplo foi capaz de se adaptar às mudanças dos parâmetros do sistema, como se pode ver pelos gráficos das figuras 7.13 e 7.14.

Como era esperado o algoritmo de mínimos quadrados recursivo com fator de esquecimento exponencial é consideravelmente mais rápido que o algoritmo de mínimos quadrados recursivo com fator de esquecimento direcional. Compare-se os gráficos das figuras 7.11 e 7.14.

7.3 Análise temporal do controlador

Com o objetivo de avaliar o comportamento temporal do bloco *AdaptCtrl* foram registados valores de latência de execução para o caso em que é utilizado esquecimento direcional e para o caso em que é utilizado o esquecimento exponencial. A plataforma utilizada neste ensaio foi:

Componente	Modelo
Processador	Intel Pentium(R) 4 CPU 3.00GHz
Gráfica	GeForce4 MX 440 AGP 8x
Memória	1GB
Motherboard	ASUS P4P800S-X (Intel ICH5/ICH5R chipset)
Sistema Operativo	Ubuntu 10.04 com kernel Linux 2.6.32-35

Na figura 7.15 podem ver-se os gráficos de frequência relativa da latência de execução para os dois casos descritos.

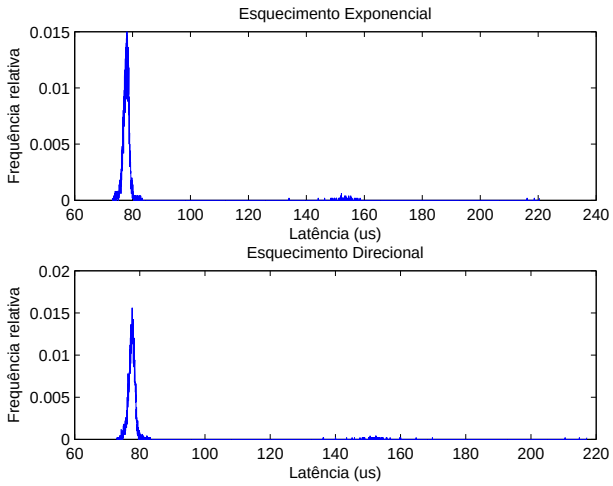


Figura 7.15: Latência de execução de *AdaptCtrl*

A tabela 7.1 apresenta a caracterização da latência de execução do bloco.

	Média (µs)	Desvio Padrão (µs)	Máximo (µs)
Esquecimento Exponencial	78.85	9.31	220.45
Esquecimento Direcional	78.61	9.23	217.11

Tabela 7.1: Caracterização da latência de execução

Os valores de latência abaixo dos quais a execução do controlador se conclui em 99% das ocasiões são apresentados na tabela 7.2.

	Exponencial	Direcional
Latência (µs)	152.07	150.90

Tabela 7.2: Valores de latência para os quais a frequência relativa cumulativa é 99%

Dos dados da tabela 7.1 conclui-se que não existem diferenças notáveis na latência de execução para os dois casos. Os valores da tabela 7.2 são de especial interesse para o utilizador quando este define o período de amostragem do sistema de controlo no *Xenomai Lab*.

Capítulo 8

Conclusões e trabalho futuro

Esta dissertação tinha dois objetivos principais: o desenvolvimento de um controlador adaptativo baseado em posicionamento de polos e o desenvolvimento de uma plataforma de ensaio constituída por um simulador de processos contínuos e a interface A-D e D-A.

O simulador de processos contínuos é capaz de simular uma variedade de sistemas, bastando ao utilizador configurar o sistema a simular através do ajuste dos potenciómetros presentes no circuito.

A interface A-D e D-A é capaz de comunicar com o PC pelos dois métodos propostos, porta paralela e porta série virtual (porta USB). É capaz de "ler" e "escrever" sinais analógicos entre os 5V e os -5V. É também capaz de operar em sistemas de controlo com frequência de amostragem superior a 2KHz.

Foram testados diversos algoritmos de identificação de sistemas na fase de pesquisa, tendo-se optado pelos métodos de mínimos quadrados recursivo com fator de esquecimento direcional e com fator de esquecimento exponencial. O utilizador tem assim a possibilidade de optar entre um algoritmo mais robusto (esquecimento direcional) e um algoritmo mais rápido na convergência das estimativas (esquecimento exponencial).

Para a resolução da equação de Diophantine estudaram-se dois métodos, um pela via polinomial (algoritmo de Kucera) e outro pela via matricial (referenciados em 4.2). De entre os dois métodos, o da via polinomial é o mais vantajoso, quer pela sua eficiência quer pela robustez. No entanto, a resolução pela via matricial é a mais simples de implementar e por isso acabou por ser a escolhida para a implementação do controlador.

A implementação dos algoritmos beneficiou bastante em termos de simplicidade do desenvolvimento da biblioteca *mtrx*. Esta biblioteca implementa uma estrutura capaz de representar matrizes e diversos métodos numéricos para cálculo matricial.

8.1 Trabalho futuro

Conforme foi demonstrado nos capítulos 4, 5, 6 e 7 obtiveram-se resultados satisfatórios para trabalho realizado. No entanto, estes resultados poderão ser melhorados.

A interface A-D e D-A beneficiaria bastante com:

- A identificação do motivo pelo qual a porta paralela apresenta valores maiores de piores casos na transmissão de alguns comandos (problema identificado no capítulo 5).

- A adição de opções de comunicação. Uma possibilidade é a comunicação pela porta *Ethernet*, sendo, no entanto, necessário desenvolver *drivers* adequados para garantir um bom comportamento temporal.
- A melhoria do circuito de acondicionamento, procurando melhorar a relação sinal-ruído.
- A adição da capacidade de auto-calibração dos circuitos de acondicionamento. Isto garantiria que a interface não adicionaria erros de ganho ou de *offset* nos dados recolhidos.
- A adição de filtros *anti-aliasing* configuráveis a partir do PC. Isto permitiria obter uma filtragem adequada em função da frequência de amostragem definida.

Por sua vez, o simulador de processos contínuos beneficiaria também com:

- A substituição dos potenciómetros por *digipots*. Isto permitiria uma maior exatidão na configuração do sistema a simular, devido à precisão dos *digipots*.
- O projeto de um novo circuito de forma a possibilitar o posicionamento dos polos do sistema a simular na parte real positiva do plano complexo do domínio de Laplace.

O controlador adaptativo pode também ser melhorado ao:

- Investigar e melhorar a filtragem dos regressores. O filtro implementado é extremamente simples e o seu numerador talvez não seja o melhor.
- Investigar e melhorar algoritmos de identificação. O controlador beneficiaria de um estimador que oferecesse um melhor compromisso entre a velocidade de convergência, exatidão das estimativas e a robustez do algoritmo.

Os algoritmos algébricos utilizados nesta dissertação são extremamente simples e possivelmente pouco eficientes, consequentemente seria possível melhorá-los ao:

- Investigar métodos numéricos e implementar algoritmos mais rápidos e eficientes que os implementados atualmente na biblioteca *mtrx*.
- Implementar uma biblioteca de manipulação de polinómios que permitisse a implementação do algoritmo de Kucera.

Apêndice A

Biblioteca de matrizes para C

O cálculo matricial é uma constante num sistema de controlo moderno. No entanto, a inexistência de um suporte direto para calculo matricial em C leva a que código escrito nesta linguagem se torne pouco legível e consequentemente aumente a probabilidade de erros de programação.

Como resposta a este problema surge a biblioteca de matrizes para C *mtrx*, desenvolvida por Diego Mendes e Jorge Azevedo. Este anexo é uma breve introdução à biblioteca, o que a compõe e como se usa. Há que ter em atenção que, apesar de funcional, esta biblioteca é ainda *work-in-progress* e alterações poderão surgir no futuro.

A.1 Mtrx

A.1.1 Estrutura, o ponto de partida

A primeira coisa a definir numa biblioteca de matrizes é a matriz em si. Assim, seguindo a linha de ferramentas como **Matlab**, definiu-se uma estrutura que não contém apenas a matriz propriamente dita, como também contém informação sobre as suas dimensões, ou seja, o seu número de linhas e colunas. A essa estrutura deu-se o nome de **Matrix** (Algoritmo A.1).

Algoritmo A.1 Estrutura de variável **Matrix**

```
1 typedef struct {
2     unsigned short rows;
3     unsigned short columns;
4     double matrix[RMAX][CMAX];
5 }Matrix;
```

Devido a questões de implementação e performance em sistemas operativos de tempo real todas as variáveis **Matrix** são inicializadas em memória estática com **RMAX** linhas e **CMAX** colunas, caso contrário o sistema poderá sofrer uma degradação de performance no que se refere a *deadlines* e *jitter* devido a imprevisibilidade do tempo necessário para que uma alocação dinâmica seja concluída. Mesmo que em memória cada variável tenha alocado espaço suficiente para uma matriz de **RMAX** linhas e **CMAX** colunas, o espaço útil está limitado para as funções de manipulação de matrizes pelas variáveis internas **rows** e **columns**.

A.1.2 Criar matrizes

Definida a estrutura, é altura de expor as diversas formas de inicializar uma matriz.

Uma das formas de inicializar uma matriz é a função **empty_matrix**. Esta função, cujo o protótipo se apresenta na linha 1 do Algoritmo A.2, é como o nome indica uma função para inicializar uma matriz de **rows** linhas e **col** colunas com todos os elementos iniciados a zero. A título de exemplo podemos ver na linha 3 de Algoritmo A.2 como iniciar uma matriz de 3 linhas e 2 colunas.

Algoritmo A.2 Função **empty_matrix**

```
1 Matrix empty_matrix(int rows,int col);
2 ...
3 Matrix Example = empty_matrix(3, 2);
```

A função **matrix_eye** é mais uma das formas de iniciar uma matriz de **dim** linhas e **dim** colunas. Esta função inicializa uma matriz quadradas com todos os elementos, menos os da diagonal principal, iniciados a zero. Aos elementos da diagonal principal é atribuído o valor **value**. A título de exemplo o resultado da linha 3 do Algoritmo A.3 é a matriz $\begin{vmatrix} 1.23 & 0 & 0 \\ 0 & 1.23 & 0 \\ 0 & 0 & 1.23 \end{vmatrix}$.

Algoritmo A.3 Função **matrix_eye**

```
1 Matrix matrix_eye(unsigned char dim, double value);
2 ...
3 Matrix Example = matrix_eye(3, 1.23);
```

A última forma de definir uma matriz é a função **new_matrix**, esta função foi pensada para ser integrada em interfaces de utilizador e é uma forma de definir uma matriz usando uma sintaxe similar à sintaxe usada por ferramentas como o Matlab. Se quisermos uma matriz igual a $\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$ devemos executar a linha 3 do Algoritmo A.4.

Algoritmo A.4 Função **new_matrix**

```
1 Matrix new_matrix(char *str);
2 ...
3 Matrix Example = new_matrix("[1,2,3;4,5,6;7,8,9]");
```

As formas de inicializar variáveis **Matrix** disponíveis nesta biblioteca cobrem, do ponto de vista dos autores, a maioria das necessidades de um utilizador comum.

A.1.3 Operações matemáticas básicas sobre matrizes

Quando se trabalha com matrizes há um conjunto de operações matemáticas essenciais. Neste biblioteca o conjunto referido é composto pelas funções apresentadas no Algoritmo A.5.

Algoritmo A.5 Funções de operações matemáticas básicas

```
1 Matrix matrix_sum(Matrix *M1, Matrix *M2);
2 Matrix matrix_sub(Matrix *M1, Matrix *M2);
3 Matrix matrix_mul(Matrix *M1, Matrix *M2);
4 Matrix matrix_mul_double(Matrix *M1, double num);
```

A função **matrix_sum** devolve a matriz resultante da soma das matrizes **M1** e **M2**.

A função **matrix_sub** devolve a matriz resultante da subtração das matrizes **M1** e **M2** (**M1**-**M2**).

A função **matrix_mul** devolve o produto das matrizes **M1** e **M2** (**M1*M2**).

A função **matrix_mul_double** devolve uma matriz com os elementos de **M1** multiplicados por **mul**.

A.1.4 Operações exclusivamente matriciais

O conjunto de operações exclusivamente matriciais presentes nesta biblioteca é composto pelas funções apresentadas no Algoritmo A.6.

Algoritmo A.6 Funções de operações matriciais

```
1 Matrix matrix_tran(Matrix *Msrc);
2 double matrix_det(Matrix *Msrc);
3 Matrix matrix_inv(Matrix *Msrc);
```

A função **matrix_tran** devolve a transposta da matriz **Msrc**.

A função **matrix_det** devolve o determinante da matriz **Msrc**.

A função **matrix_inv** devolve a matriz inversa da matriz **Msrc**, quando a mesma é possível de calcular.

A.1.5 Representação de matrizes

Para a representação de matrizes em consola ou escrita das mesmas em ficheiros é apresentada em Algoritmo A.7 a função **matrix_print**.

Algoritmo A.7 Função **matrix_print**

```
1 void matrix_print(Matrix * M1);
```

Apêndice B

How to build a simple serial bootloader for PIC32

This guide is aimed for anyone looking to build a simple bootloader for a generic development board based on the PIC32MX Family. Still, the guide itself is oriented for the DETPIC32 from the University of Aveiro.

This guide is essentially a simplification of the application note AN1388 from Ganapathi Ramachandra available on the Microchip website[1].

B.1 Prerequisites

For this guide it is required that the user has a version of the MPLAB X IDE and the C32 compiler installed.

If this is not the case, the user can get the required installers as well as the required installation instructions in:

`http://ww1.microchip.com/downloads/mplab/X/index.html`

B.2 Preparing the Bootloader

1. Download the source code from:

   `ww1.microchip.com/downloads/en/AppNotes/AN1388_Source_Code_011112.zip`

2. Extract and execute the Universal Bootloader executable.

3. In the installation folder we have 2 different folders, one with the firmware for the PIC32 and another with the GUI to interact with the bootloader. Execute MPLAB X and open the UART_Btl_Explorer16.X project inside "Firmware\Bootloader\MPLAB_X_Workspace".

4. Let's now start by preparing the bootloader.

5. Start by defining the desired clocks for the PIC32 by configuring the following pragma entries on "bootloader.c" (details can be found on Figure 6-1 (PIC32 Family OscillatorSystem Block Diagram) of the PIC32 Family Reference Manual, Sect. 06 Oscillators):

[1] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en554836

- FPLLIDIV(CPU clock)
- FPLLMUL(CPU clock)
- FPLLODIV(CPU clock)
- FPBDIV(Peripheral clock)

6. We want the application to be placed into the "User Application Code and IVT" area[2], which is placed between 0x9D000000 and 0x9D07FFFF. So open the file "bootloader.h" and edit the values for the following MACROS:

```
#define APP_FLASH_BASE_ADDRESS 0x9D000000
#define APP_FLASH_END_ADDRESS 0x9D07FFFF
#define USER_APP_RESET_ADDRESS (0x9D000000 + 0x1000 + 0x970)
```

7. Because the bootloader as it is depends on a switch to enter the "Firmware Upgrade Mode" and such a switch is not present in the DETPIC32 board, we need to alter the software structure in a way that it no longer requires the refereed switch. There's also the UART connection that comes defined for the UART2 module and needs to be changed.

8. Let's start by the UART. For the DETPIC32 board we must configure the bootloader to access the UART1, so open the header "Uart.h" and change the line

```
#define Ux(y) U2##y
```

to

```
#define Ux(y) U1##y
```

9. Now for the main code I suggest the following.

```
INT main(void) {

  UINT pbClk;

  /* Setup configuration */
  pbClk = SYSTEMConfig(SYS_FREQ, SYS_CFG_WAIT_STATES | SYS_CFG_PCACHE);

  /* Initialize the transport layer - UART/USB/Ethernet */
  TRANS_LAYER_Init(pbClk);

  /* Stays in the firmware upgrade mode while no timeout
  occurs or if no application is present or if  the GUI
  is connected to the MCU */
  while ((ReadCoreTimer() < 0x1312D00) || WORKING() || !ValidAppPresent()) {
    /* Enter firmware upgrade mode. */
    /* Be in loop, looking for commands from PC */
    TRANS_LAYER_Task(); /* Run Transport layer tasks */
    FRAMEWORK_FrameWorkTask(); /* Run frame work related
    tasks (Handling Rx frame, process frame and so on) */
  }
```

[2] AN1388 FIGURE 1, pag. 2

```
  /* Close trasnport layer. */
  TRANS_LAYER_Close();

  /* No trigger + valid application = run application. */
  JumpToApp();

  return 0;
}
```

Where the condition (ReadCoreTimer() < 0x1312D00) is set for the bootloader to wait for 1 second before moving on to the application, if there is a valid one. If you wish to have a different waiting time you only need to change the value in this condition, bear in mind that the Core Timer counter is incremented every two periods of the CPU clock[3].

The WORKING() function solves a specific problem, if there's already a valid application and the GUI tries to establish contact with the bootloader it must perform all the desired actions before the timeout occurs, otherwise the bootloader will leave the "Firmware Upgrade Mode" before finishing what the user wants.

So if we use WORKING() and make it start returning TRUE once the connection between the GUI and the bootloader is established, the bootloader will stay inside the loop waiting for the rest of the commands from the PC. The only way the bootloader leaves that loop is if it receives a "JMP_TO_APP" command from the GUI or the PIC32 is reset.

To make this possible we must open the "Framework.c" file and add the following global variable and function:

```
static BOOL Working = FALSE;
...
BOOL WORKING(void){
    return Working;
}
```

The next step consists in editing the "HandleCommand" function on the same file. Start by editing the case "READ_BOOT_INFO"(this request is sent every time the GUI tries to establish a connection to the bootloader so we use this to our favor).

```
...
case READ_BOOT_INFO://Read boot loader version  info.
  memcpy(&TxBuff.Data[1], BootInfo,2);
  //Set the transmit frame length.
  TxBuff.Len = 2 + 1;//Boot Info Fields + command
  Working=1;
  break;
...
```

And finally in the "JMP_TO_APP" case.

```
...
```

[3]http://blog.flyingpic24.com/2009/04/02/using-the-32-bit-core-timer/

```
case JMP_TO_APP:
  // Exit firmware upgrade mode.
  RunApplication = TRUE;
  Working=0;
  break;
...
```

10. The bootloader is now ready. Compile it and use your programmer of choice to load it into the PIC32.

B.3 Preparing the application's linker

1. Right click the project » New » Empty File and create a *.ld file with the name you prefer(ex: linker.ld)

2. Copy the contents of "\pic32mx\lib\ldscripts\elf32pic32mx.x" to the newly created file. The folders are located in the C32 compiler installation folder.

3. Replace the "INCLUDE procdefs.ld" directive with

   "INCLUDE ..\lib\proc\32MX795F512H\procdefs.ld".

4. Copy the following lines of code and paste them after the PROVIDE directives and before "SECTIONS".

```
/*****************************************************************
* Processor-specific object file. Contains SFR definitions.
*****************************************************************/
INPUT("processor.o")
/*****************************************************************
* For interrupt vector handling
*****************************************************************/
PROVIDE(_vector_spacing = 0x00000001);
/* _ebase_address value must be same as the ORIGIN value of exception_mem
(see below) */
_ebase_address = 0x9D000000;
/*****************************************************************
* Memory Address Equates
*****************************************************************/
/* Equate _RESET_ADDR to the ORIGIN value of kseg1_boot_mem (see below) */
_RESET_ADDR = (0x9D000000 + 0x1000 + 0x970);
/*Map _BEV_EXCPT_ADDR and _DBG_EXCPT_ADDR in to kseg1_boot_mem (see below) */
/* Place _BEV_EXCPT_ADDR at an offset of 0x380 to _RESET_ADDR */
/* Place _DBG_EXCPT_ADDR at an offset of 0x480 to _RESET_ADDR */
_BEV_EXCPT_ADDR = (0x9D000000 + 0x1000 + 0x970 + 0x380);
_DBG_EXCPT_ADDR = (0x9D000000 + 0x1000 + 0x970 + 0x480);
_DBG_CODE_ADDR = 0xBFC02000;
_DBG_CODE_SIZE = 0xFF0 ;
_GEN_EXCPT_ADDR = _ebase_address + 0x180;
/*****************************************************************
* Memory Regions
*
```

```
 * Memory regions without attributes cannot be used for orphaned sections.
 * Only sections specifically assigned to these regions can be allocated
 * into these regions.
 *********************************************************************/
MEMORY{
/* IVT is mapped into the exception_mem. ORIGIN value of exception_mem must
align with 4K address boundary. Keep the default value for the length */
exception_mem : ORIGIN = 0x9D000000, LENGTH = 0x1000
/* Place kseg0_boot_mem adjacent to exception_mem. Keep the default value
for the length */
kseg0_boot_mem : ORIGIN = (0x9D000000 + 0x1000), LENGTH = 0x970
/* C Start-up code is mapped into kseg1_boot_mem. Place kseg1_boot_mem
adjacent to
kseg0_boot_mem. Keep the default value for the length */
kseg1_boot_mem : ORIGIN = (0x9D000000 + 0x1000 + 0x970), LENGTH = 0x490
/*All C files (Text and Data) are mapped into kseg0_program_mem. Place
kseg0_program_mem adjacent to kseg1_boot_mem. Change the length of
kseg0_program_mem as required. In this example, 512 KB Flash size
is shrunk as follows: */
kseg0_program_mem (rx):ORIGIN = (0x9D000000 + 0x1000 + 0x970 + 0x490),
LENGTH = (0x80000 - (0x1000 + 0x970 + 0x490))
debug_exec_mem : ORIGIN = 0xBFC02000, LENGTH = 0xFF0
config3 : ORIGIN = 0xBFC02FF0, LENGTH = 0x4
config2 : ORIGIN = 0xBFC02FF4, LENGTH = 0x4
config1 : ORIGIN = 0xBFC02FF8, LENGTH = 0x4
config0 : ORIGIN = 0xBFC02FFC, LENGTH = 0x4
kseg1_data_mem (w!x) : ORIGIN = 0xA0000000, LENGTH = 0x20000
sfrs : ORIGIN = 0xBF800000, LENGTH = 0x100000
configsfrs : ORIGIN = 0xBFC02FF0, LENGTH = 0x10
}
```

5. Finally, clean and build the project and use the GUI to load your application to the bootloader equipped DETPIC32 board.

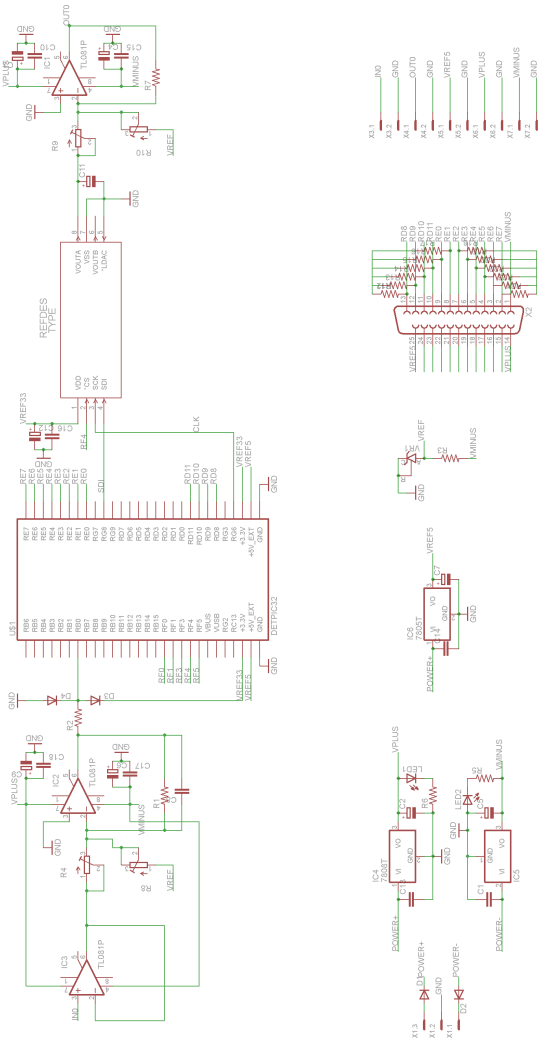
6. For every application you want to load onto the DETPIC32, you must add this linker to its project.

B.4 Closing remarks

By now you should have a functioning bootloader and the required linker for your projects. If you wish to use these files in different development boards the changes required are minimal so it should easy, still reading the AN1388 is advised.

Apêndice C

Esquemático de Interface A-D e D-A



Apêndice D

Resolução da equação de Diophantine (Código Matlab)

D.1 Código do ensaio

```
clc
close all
clear all
B=[0 0.009 0.007];
A=[1 -1.724 0.741];
Am=[1 -1.5 0.7];
Ao=[1 0];
Rd=[1];
Sd=[1];
N=50000;
t1=zeros(1,N);
t2=zeros(1,N);
for i=1:N
x=tic;
[R,S,T]=synth(A,Rd,B,Sd,Am,Ao);
t1(i)=toc(x);
end
B=B(2:end);
for i=1:N
x=tic;
[R,S,T]=synth_poly(A,Rd,B,Sd,Am,Ao);
t2(i)=toc(x);
end
[y1,x1]=hist(t1,5000);
[y2,x2]=hist(t2,5000);
figure(1)
plot(1e6*x1,y1/N,'b--',1e6*x2,y2/N,'r-') legend('Matricial','Kucera');
xlabel('Latencia (us)');
ylabel('Probabilidade');
```

```
avg1=mean(t1)
avg2=mean(t2)
std1=std(t1)
std2=std(t2)
```

D.1.1 Função synth

```
function [R,S,T]=synth(A,Rd,B,Sd,Am,Ao)

if length(Ao)~=2*(length(A)-1)+(length(Rd)-1)+(length(Sd)-1)-(length(Am)-1)
    error('deg(Ao)~=2deg(A)+deg(Rd)+deg(Sd)-deg(Am)-1');
end

Ac=conv(Am,Ao);
Ax=conv(A,Rd);
Bx=conv(B,Sd);

degRx=(length(A)-1)+(length(Sd)-1)-1;
degSx=(length(A)-1)+(length(Rd)-1)-1;
Rx=zeros(1,degRx+1);
Sx=zeros(1,degSx+1);

%Builds the Sylvester matrix for polynomials Ax and Bx
sylvester=mat_sylv(Ax,Bx,degRx+1,degSx+1);

if det(sylvester)<0.000001
    error('Polynomials A and B have common factors');
end

%Returns the solution to AxRx+BxSx=Ac
RS=mat_solve(sylvester,Ac);

for i=1:degRx+1
    Rx(i)=RS(i);
end
for i=1:degSx+1
   Sx(i)=RS(degRx+1+i);
end

R=conv(Rx,Rd);
S=conv(Sx,Sd);
T=Ao*sum(Am)/sum(B);
end
```

D.1.1.1 Função mat_sylv

```
function Sylv=mat_sylv(A,B,colA,colB)
```

```
Sylv=zeros(colA+colB,colA+colB);
for i=1:colA
   for j=1:length(A)
       Sylv(j+i-1,i)=A(j);
   end
end
for i=1:colB
   for j=1:length(B)
        Sylv(j+i-1,colA+i)=B(j);
   end
end
end
```

D.1.1.2 Função mat_solve

```
function B=mat_solve(A,C)
% A*B=C
% For a known matrix A and a known vector C, calculates vector B

[m,n]=size(A);
if m~=n
    error('MAT_SOLVE : Number of equations differ from number of variables');
end
if m~=length(C)
    error('MAT_SOLVE : Vector C length and Matrix A number of rows differ');
end

aux=zeros(m,n+1);
C=C(:);
aux(1:m,1:n)=A;
aux(1:m,n+1)=C;

for i=1:m
    aux(i,:)=aux(i,:)/aux(i,i);
    for j=1:m
        if j~=i
            aux(j,:)=aux(j,:)-aux(j,i)*aux(i,:);
        end
    end
end

B=aux(:,n+1)';
```

D.1.2 Função synth_poly

```
function [R,S,T]=synth_poly(A,Rd,B,Sd,Am,Ao)
```

```
if length(Ao)~=2*(length(A)-1)+(length(Rd)-1)+(length(Sd)-1)-(length(Am)-1)
    error('deg(Ao)~=2deg(A)+deg(Rd)+deg(Sd)-deg(Am)-1');
end

Ac=conv(Am,Ao);
Ax=conv(A,Rd);
Bx=conv(B,Sd);
Ac2=zeros(1,length(Ac));
Ax2=zeros(1,length(Ax));
Bx2=zeros(1,length(Bx));
for i=1:length(Ac2);
   Ac2(i)=Ac(end-i+1);
end
for i=1:length(Ax2);
   Ax2(i)=Ax(end-i+1);
end
for i=1:length(Bx2);
   Bx2(i)=Bx(end-i+1);
end

[Rmin,Smin]=Kucera(Ax2,Bx2,Ac2);

Rmin2=zeros(1,length(Rmin));
Smin2=zeros(1,length(Smin));

for i=1:length(Rmin)
    Rmin2(i)=Rmin(end-i+1);
end
for i=1:length(Smin)
    Smin2(i)=Smin(end-i+1);
end
R=conv(Rmin2,Rd);
S=conv(Smin2,Sd);
T=Ao*sum(Am)/sum(B);
end
```

D.1.2.1 Função Kucera

```
function [Rmin,Smin]=Kucera(A,B,Ac)
nA=length(A);
nB=length(B);
nG=nA;
nR=nB;
G=zeros(1,nG);
R=zeros(1,nG);
for i=1:nG
    G(i)=A(i);
```

```
end
for i=1:nR
    R(i)=B(i);
end
n=nG;
X=zeros(1,2*n-1);
X(1)=1;
Y=zeros(1,2*n-1);
U=zeros(1,2*n-1);
V=zeros(1,2*n-1);
V(1)=1;
while (nR>1) || (abs(R(1))>0.001)
    delta=nG-nR;
    if(delta > 0)
        quo=G(nG)/R(nR);
        for i=1:1:n-delta
            G(delta+i)=G(delta+i)-quo*R(i);
            X(delta+i)=X(delta+i)-quo*U(i);
            Y(delta+i)=Y(delta+i)-quo*V(i);
        end
        while abs(G(nG))<0.001
            nG=nG-1;
        end
    else
        quo=R(nR)/G(nG);
        for i=1:1:n+delta
            R(-delta+i)=R(-delta+i)-quo*G(i);
            U(-delta+i)=U(-delta+i)-quo*X(i);
            V(-delta+i)=V(-delta+i)-quo*Y(i);
        end
        while (abs(R(nR))<0.001)&&(nR>1)
            nR=nR-1;
        end
    end
end
nX=2*n-1;
nY=nX;
nU=nX;
nV=nX;
while (nX>1)&& (abs(X(nX))<0.001)
    nX=nX-1;
end
while (nY>1)&& (abs(Y(nY))<0.001)
    nY=nY-1;
end
while (nU>1)&& (abs(U(nU))<0.001)
    nU=nU-1;
```

```
end
while (nV>1)&& (abs(V(nV))<0.001)
    nV=nV-1;
end

%X Ac div G
%Q= Ac div G
nAc=length(Ac);
if(nAc>nG)
    Q=zeros(1,nAc-nG+1);
    Resto=Ac;
    nQ=nAc-nG+1;
    for i=1:nQ
        Q(nQ-i+1)=Resto(nAc-i+1)/G(nG);
        for k=1:nG
            Resto(nAc-i-k+2)=Resto(nAc-i-k+2)-G(nG-k+1)*Q(nQ-i+1);
        end
    end
end
%R0=X*Ac div G
R0=zeros(1,nX+length(Q)-1);
for i=1:nX
    for k=1:length(Q)
        R0(i+k-1)=R0(i+k-1)+X(i)*Q(k);
    end
end
%S0=Y*Ac div G
nS0=nY+length(Q)-1;
S0=zeros(1,nS0);
for i=1:nY
    for k=1:length(Q)
        S0(i+k-1)=S0(i+k-1)+Y(i)*Q(k);
    end
end
%Q= S0 div V
if(nS0>nV)
    Q=zeros(1,nS0-nV+1);
    Smin=S0;
    nQ=nS0-nV+1;
    for i=1:nQ
        Q(nQ-i+1)=Smin(nS0-i+1)/V(nV);
        for k=1:nV
            Smin(nS0-i-k+2)=Smin(nS0-i-k+2)-V(nV-k+1)*Q(nQ-i+1);
        end
    end
end
```

```
nAux=(nQ+nU-1);
Aux=zeros(1,nAux);

for i=1:nQ
    for k=1:nU
        Aux(i+k-1)=Aux(i+k-1)+Q(i)*U(k);
    end
end

%Rmin=R0+(-S0 div V)*U
nRmin=max((nQ+nU-1),(nX+length(Q)-1));
Rmin=zeros(1,nRmin);

for i=1:nX+length(Q)-1
    Rmin(i)=R0(i);
end
for i=1:nQ+nU-1
    Rmin(i)=Rmin(i)-Aux(i);
end

while Rmin(end)<0.0001
    Rmin=Rmin(1:end-1);
end
while Smin(end)<0.0001
    Smin=Smin(1:end-1);
end

end
```

Bibliografia

[1] Karl J. Åström and Björn Wittenmark. *Computer Controlled Systems: Theory and Design*. Prentice-Hall, 1997.

[2] Karl J. Åström and Björn Wittenmark. *Adaptive Control*. Dover, 2nd edition, 2008.

[3] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *IEEE Computer*, 21(10):10–19, Outubro 1988.

[4] Novembro 2012 2012. URL `http://www.lynuxworks.com/rtos/rtos.php`.

[5] Novembro 2012. URL `http://www.qnx.com/`.

[6] Novembro 2012. URL `http://www.windriver.com/products/vxworks/`.

[7] Novembro 2012. URL `http://www.xenomai.org/`.

[8] Jorge Manuel Coelho Amado de Azevedo. Xenomai lab - a platform for digital real-time control. Master's thesis, Universidade de Aveiro - Departamento de Electrónica, Telecomunicações e Informática., 2011.

[9] Novembro 2012. URL `http://jorgeazevedo.github.com/xenomai-lab/`.

[10] Novembro 2012. URL `http://www.mathworks.com/products/simulink/`.

[11] Novembro 2012. URL `http://www.scilab.org/products/xcos`.

[12] Katsuhiko Ogata. *Discrete-Time Control Systems*. Prentice-Hall International, Inc., 2nd edition, 1995.

[13] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[14] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing: Principles, Algorithms and Applications*. Prentice-Hall, Inc., 3rd edition, 1996.

[15] Wei Yu, David I. Wilson, Jonathan Currie, and Brent R. Young. The robustness of pi & pid controllers in the presence of sampling jitter. In *Chemical Process Control VIII, FOCAPO/CPC 2012*, 11–13 January 2012.

[16] Jan Jezek. New algorithm for minimal solution of linear polynomial equations. *Kybernetika*, 18(6):505–516, 1982.

[17] Selecting a model structure in the system identification process, Setembro 2012. URL http://www.ni.com/white-paper/4028/en.

[18] R. Brincker, L. Zhang, and P. Andersen. Modal identification from ambient response using frequency domain decomposition. In *18th International Modal Analysis Conference*, San Antonio, Texas, February 7-10 2000.

[19] Alexandre Manuel Mota. Identificação de sistemas. 2000.

[20] Liyu Cao and Howard Schwartz. A directional forgetting algorithm based on the decomposition of the information matrix. *Automatica*, 36:1725–1731, 2000.

[21] P. E. Wellstead and M. B. Zarrop. *Self-Tuning Systems: Control and Signal Processing*. Wiley, 1991.

[22] Eric W. Weisstein. "gaussian elimination". URL http://mathworld.wolfram.com/GaussianElimination.html.

[23] Julho 2012. URL http://www.physicsforums.com/showthread.php?t=6398.

[24] Julho 2012. URL http://www.zelscope.com/faq.html.

[25] Julho 2012. URL http://www.ledametrix.com/oscope/index.html.

[26] Julho 2012. URL http://www.ni.com/data-acquisition/multifunction/.

[27] Julho 2012. URL http://sine.ni.com/nips/cds/view/p/lang/en/nid/201612.

[28] Julho 2012. URL http://www.ieeta.pt/~jla/.

[29] Media John Garney and Intel Architecture Labs Interconnect Technology. An analysis of throughput characteristics of universal serial bus. page 1.

[30] Future Technology Devices International Ltd. Mm232r usb - serial uart development module datasheet.

[31] Henry Spencer. Simple serial protocol(ssp).

[32] Novembro 2012. URL http://www.ieeta.pt/~jla/ac2/Labs/DETPIC32.pdf.

[33] Julho 2012. URL http://www.ftdichip.com/FTDrivers.htm.

[34] Novembro 2012. URL http://retired.beyondlogic.org/spp/parallel.pdf.

[35] Julho 2012. URL http://parapin.sourceforge.net/.

[36] Frank D. Gage Vannevar Bush and Herbert R. Stewart. A continuous intergraph. *Journal of the Franklin Institute*, 208:63–84, 1927.

[37] Vannevar Bush Harold L. Hazen. Intergraph solutions of differential equations. *Journal of the Franklin Institute*, 208:575–615, 1927.

[38] Jerome B. Wiesner. *Vannevar Bush 1890-1974: A Biographical Memoir*. National Academy of Sciences, 1979.

[39] Vannevar Bush. The differential analyzer. a new machine for solving differential equations. *Journal of the Franklin Institute*, 212:447–488, 1931.

[40] A. B. MacNee. An electronic differential analyzer. *Proceedings of the IRE*, pages 17–20, November 1949.

[41] Robert M. Howe. Analog computers in academia and industry. *IEEE Control Systems Magazine*, pages 38–39, June 2005.

[42] T. R. Kennedy. Electronic computer flashes answers, may speed engineering. *New York Times*, February,15 1946.

[43] Ray Spiess. The comdyna gp-6 analog computer: Alive but not exactly kicking. *IEEE Control Systems Magazine*, pages 68–73, June 2005.

[44] Ray Spiess. Two reasons why a controls laboratory needs an analog computer. In *American Control Conference*, Seattle, WA, USA, 18-20 June 1986.

[45] Andrew L. Fitch, Herbert H. C. Iu, and Dylan D. C. Lu. An analogue computer for electronic engineering education. *IEEE Transactions on Education*, 54:550–557, 2011.